



# User Guide

# ARCAD Transformer

# Field

Version 23.0

**Publication Date: January, 2023**

*Prepared by the ARCAD Documentation Team*



#### North America & LATAM

1 N. State St, 15th Floor  
Chicago, IL  
USA  
1-603-371-9074  
1-603-371-3256 (support calls only)  
sales-us@arcadsoftware.com

#### EMEA (HQ)

55 Rue Adrastée – Parc Altaïs  
74650 Chavanod/Annecy  
France  
+33 450 578 396  
sales-eu@arcadsoftware.com

#### Asia Pacific

5 Shenton Way #22-04  
UIC Building  
Singapore 068808  
sales-asia@arcadsoftware.com

**Copyright © 1992-2023 by ARCAD. All rights reserved.**

The following terms are names owned by International Business Machines Corporation in the United States, other countries, or both: AS/400®, ClearCase, ClearQuest®, DB2, DB2 Connect™, DB2 Universal Database™, ibm.com, IBM i, iSeries, System i, OS/400, Rational®, SP2, Service Pack, WebSphere. Java and all names based on Java are owned by Oracle Corp. in the United States, other countries, or both. Eclipse is a registered trademark of Eclipse Foundation, Inc. Other names of companies, products or services are the property of their respective owners.

## Contact ARCAD

Headquartered in France at the foot of the Alps, ARCAD offers global services and has offices and partners all over the world. ARCAD partners with leading-edge companies throughout the world to offer full services, close to home.

Visit our website to [Contact Us](#) and find out more about our company and partners, or to request a demo.

The [ARCAD Customer Portal](#) is intended for current and potential customers that have full or trial versions of ARCAD software. If you already use or are interested in using an ARCAD product, the portal lets you view all of your current licenses and generate your own temporary license keys for most ARCAD products. It grants you access to the ARCAD product knowledge base (new releases, release notes and current documentation).

Do you have a request for change or have you encountered a bug? Log into the [ARCAD Helpdesk](#) and create a ticket.

ARCAD guarantees consultant support 24 hours a day, 5 days a week (24/5) to registered members. Calls received are redirected, according to the hour, to put you in contact with a support team in or near your timezone.

| Country   | Address   | Account Contact   | Support Contact  |
|-----------|---|---|--|
| France    | ARCAD Software (HQ)<br>55 Rue Adrastée<br>74650 Chavanod  | +33 4 50 57 83 96<br><a href="mailto:sales-eu@arcadsoftware.com">sales-eu@arcadsoftware.com</a>   | Worldwide 24/7:<br>+1 603 371 3256<br><br>France only:<br>+33 450 57 28 00<br><br><a href="mailto:support@arcadsoftware.com">support@arcadsoftware.com</a><br><br><a href="#">ARCAD Helpdesk</a> |
|           | ARCAD Software<br>17 chemin de la plaine<br>07200, Saint-Didier-sous-Aubenas                                |   |  |
| Germany   | ARCAD Software Deutschland GmbH<br>c/o Pramex International GmbH<br>Savignystr. 43, 60325 Frankfurt am Main |   |  |
| China     | ARCAD Software<br>#2035, Yuehai Plaza,<br>180 Wanbo 2nd Road,<br>Nancun, Panyu District, Canton             |   |  |
| India     | ARCAD Software<br>D-280/281/282, Vibhuti Khand<br>Gomti Nagar, Lucknow                                      | +86 (020)22324643<br>+86 (020)22324649<br><a href="mailto:sales-asia@arcadsoftware.com">sales-asia@arcadsoftware.com</a>                  |  |
| Singapore | ARCAD Software<br>5 Shenton Way #22-04<br>UIC Building<br>Singapore 068808                                  |   |  |
| USA       | ARCAD Software<br>1 N. State St, 15th Floor<br>Chicago, IL  | +1 (603) 371-9074<br>+1 (603)-371-3256 (support calls only)<br><a href="mailto:sales-us@arcadsoftware.com">sales-us@arcadsoftware.com</a> |  |

Table 1: Contact ARCAD

# Preface

## Document Purpose

This document is intended to guide you through using ARCAD Transformer Field.

## Intended Audience

This document is intended for all ARCAD Transformer Field users.

## Related Documentation

ARCAD technical documentation can be accessed from the product's online help or by logging into the [Customer Portal](#) on our website.

| Related Documentation |
|-----------------------|
| Release Notes         |
| ARCAD Glossary        |

Table 2: Related Documentation

Unless stated otherwise, all content is valid for the most current version of ARCAD Transformer Field listed as well as every subsequent version.

| Product Version | Document Version | Publication Date | Update Record   |
|-----------------|------------------|------------------|---|
| ≥ 23.0          | 2.5              | January, 2023    | No functional changes.  |
| 22.0            | 2.4              | January, 2022    | No functional changes.<br>Added the mandatory parameters changes to load cross-references for ARCAD Transformer Field projects. |

Table 3: ARCAD Transformer Field User Guide Publication Record

# Contents

---

|                            |           |
|----------------------------|-----------|
| <b>Contact ARCAD</b> ..... | <b>3</b>  |
| <b>Preface</b> .....       | <b>4</b>  |
| <b>Contents</b> .....      | <b>5</b>  |
| <b>Tables</b> .....        | <b>10</b> |
| <b>Figures</b> .....       | <b>11</b> |

## INTRODUCTION

---

|   |           |
|---|-----------|
| <b>1 About ARCAD Transformer Field</b> .....                        | <b>13</b> |
| 1.1 Functional constraints.....                                     | 13        |
| 1.1.1 Understand cross-reference problems.....                      | 13        |
| 1.1.2 Work around incorrect files.....                              | 13        |
| 1.1.3 Plan simultaneous processes.....                              | 14        |
| 1.1.4 Flag or modify for multi-users.....                           | 14        |
| <b>2 Pre-requisites</b> .....                                       | <b>15</b> |
| 2.1 Load the repository.....  | 15        |
| 2.2 Create a working environment.....                               | 15        |
| 2.3 Train for ARCAD.....  | 16        |
| <b>3 Main concepts</b> .....  | <b>17</b> |
| 3.1 Detailed impact analysis for unmodified fields.....             | 17        |
| 3.2 Detailed impact analysis for automatically-modified fields..... | 17        |
| 3.3 Data recovery programs.....                                     | 18        |
| 3.4 Operational chronology for data recovery.....                   | 18        |
| 3.5 Quality over speed.....   | 18        |
| <b>4 Overview of ARCAD Transformer Field</b> .....                  | <b>19</b> |

## PROPAGATION FUNCTIONS

---

|  |           |
|--|-----------|
| <b>5 Working with propagation functions</b> .....                  | <b>22</b> |
| <b>6 Identifying fields</b> .....                                  | <b>23</b> |
| <b>7 Outlining propagation behavior</b> .....                      | <b>24</b> |
| 7.1 Alphanumeric-type propagation.....                             | 24        |
| 7.2 Numeric-type propagation.....                                  | 25        |
| <b>8 Languages processed by the propagation</b> .....              | <b>27</b> |
| <b>9 Evaluating lists of impacted fields</b> .....                 | <b>28</b> |
| 9.1 Evaluating lists of physical file fields.....                  | 28        |
| 9.2 Evaluating lists of program fields.....                        | 28        |
| 9.3 Evaluating lists of display or printer fields (DSPF/PRTF)..... | 29        |

|   |           |
|---|-----------|
| 9.4 Evaluating lists of fields in the L.D.A. to propagate.....        | 29        |
| <b>10 The propagation layout.....</b>                                 | <b>31</b> |
| 10.1 Propagating lists of fields – LSTxxxx.....                       | 33        |
| 10.2 Propagating lists of programs to be processed – LSTPGM.....      | 34        |
| 10.3 Propagating lists of propagation stoppers – LSTPRPSTP.....       | 34        |
| 10.4 Propagating lists of D.D.S. fields – LSTDDSFLD.....              | 34        |
| 10.5 Propagating lists of derived fields – LSTDRVFLD.....             | 36        |
| 10.6 Propagating source line files – AARFCHSF1.....                   | 37        |
| 10.7 Propagating other AARF... files.....                             | 38        |
| 10.8 Propagating source lines.....                                    | 39        |
| 10.9 Propagating lists of other D.B. fields.....                      | 40        |
| 10.10 Propagating lists of anomalies.....                             | 41        |
| 10.10.1 Propagate lists of anomaly component fields.....              | 41        |
| 10.10.2 Propagate lists of stoppers used.....                         | 42        |
| 10.10.3 Propagate lists of components outside propagation.....        | 42        |
| 10.10.4 Propagate lists of components in source access error.....     | 42        |
| <b>11 Understanding field formats.....</b>                            | <b>44</b> |
| 11.1 Indicating field formats in lists of propagated fields.....      | 44        |
| 11.1.1 Field format contents.....                                     | 44        |
| 11.2 Field format codes.....  | 45        |
| 11.3 Compressed field formats.....                                    | 46        |
| 11.4 Extended field formats.....                                      | 46        |
| <b>12 Working with automatic modification functions.....</b>          | <b>47</b> |
| 12.1 Modifying logical files.....                                     | 48        |
| 12.2 Modifying program fields automatically.....                      | 49        |
| 12.3 Modifying display/printer fields automatically – ACVTDDSFLD..... | 51        |
| 12.3.1 Field length extension.....                                    | 52        |
| 12.3.2 Field deletion.....  | 53        |
| 12.4 Adding a file field.....   | 53        |
| 12.4.1 Associated display fields (\$\$A1).....                        | 53        |
| 12.4.2 No associated display fields (\$\$A1 is blank).....            | 54        |
| 12.4.3 Add a 3rd display/status field (\$\$F3).....                   | 54        |
| 12.5 Modifying interface file fields automatically.....               | 55        |
| <b>13 Interpreting the nature of a source line.....</b>               | <b>56</b> |

## IMPACT ANALYSIS

---

|   |           |
|---|-----------|
| <b>14 Working with field lists.....</b>                         | <b>59</b> |
| 14.1 Creating/refreshing lists of database fields – FLDDCT..... | 59        |
| 14.2 Building field lists.....                                  | 59        |
| 14.2.1 Create a field extraction macro-command.....             | 60        |

|  |    |
|--|----|
| 14.2.1.1 Review the list of fields .....                               | 61 |
| 14.2.2 Build the list of fields to propagate from ARCAD Observer ..... | 62 |
| 14.2.3 Retrieve the list of fields to process .....                    | 62 |
| 14.3 Setting the field format – ASETFLDFMT .....                       | 63 |
| 14.3.1 Set the same field format for all fields .....                  | 64 |
| 14.3.2 Set a particular format for specific fields .....               | 64 |

## TRANSFORMING

|  |            |
|--|------------|
| <b>15 Working with the TRANSFORM menu: Propagation and Transformations .....</b> | <b>67</b>  |
| <b>16 Detecting anomalies generated by propagation .....</b>                     | <b>69</b>  |
| <b>17 Propagation types – AWRKPRPTYP .....</b>                                   | <b>70</b>  |
| 17.1 Defining a propagation type .....   | 70         |
| 17.2 Identifying propagation type and behavior .....                             | 70         |
| 17.3 Defining automatic modifications at file level .....                        | 74         |
| 17.4 Defining automatic modifications at screen/printer level .....              | 80         |
| <b>18 Standard routines .....</b>  | <b>86</b>  |
| 18.1 Naming and storing standard routines .....                                  | 86         |
| 18.2 Using routines .....  | 87         |
| 18.3 Ascertaining routine particularities .....                                  | 87         |
| 18.4 Using variables in standard routines .....                                  | 88         |
| 18.5 Checking language in standard routines .....                                | 89         |
| 18.6 Inserting standard routines .....   | 91         |
| 18.6.1 Insert routines in RPG – RPGLE .....                                      | 91         |
| 18.6.2 Insert routines in CLP .....  | 92         |
| 18.6.3 Insert routines in COBOL .....  | 92         |
| 18.6.4 Insert no routines .....  | 92         |
| 18.7 Working with standard routines – AWRKSTDRTN .....                           | 92         |
| 18.7.1 Set example 1 of a standard routine (Basic) .....                         | 93         |
| 18.7.2 Set example 2 of a standard routine (Complex) .....                       | 95         |
| <b>19 Modifying D.B. files automatically – ACVTDBFFLD .....</b>                  | <b>99</b>  |
| 19.1 Adding fields .....   | 100        |
| <b>20 Propagating and modifying programs automatically – ACVTPGMFLD .....</b>    | <b>101</b> |
| <b>21 Modifying DDS automatically – ACVTDDSFLD .....</b>                         | <b>105</b> |
| <b>22 Modifying FRM – AWRKFRMCHG .....</b>                                       | <b>108</b> |
| <b>23 Printing the impact analysis – APRTRMCHG .....</b>                         | <b>110</b> |
| <b>24 Displaying derived fields – ADSPDRVFLD .....</b>                           | <b>114</b> |
| <b>25 Setting propagation stoppers – ASETPRPSTP .....</b>                        | <b>118</b> |
| 25.1 Consulting literals detected by the propagation .....                       | 121        |

|   |            |
|---|------------|
| <b>26 Applying generated modifications – AAPYFRMCHG</b> .....         | <b>123</b> |
| 26.1 Compiling the modified and dependent objects.....                | 124        |
| <b>27 Working with other options in the TRANSFORM menu</b> .....      | <b>126</b> |
| 27.1 Configuring transformer marking parameters.....                  | 126        |
| 27.2 Deleting information generated by propagation – ACLRDVRFLLD..... | 127        |
| 27.3 Converting a list of fields to a list of sources – ACVTLLST..... | 127        |

## DATA RECOVERY

---

|  |            |
|--|------------|
| <b>28 Recovering data</b> .....  | <b>129</b> |
| 28.1 Working with data recovery sequences.....                         | 129        |
| 28.1.1 Write a standard routine.....                                   | 130        |
| 28.1.2 Verify or update the impacted fields list.....                  | 130        |
| 28.1.3 Generate data field formatting programs – AGENFMTPGM.....       | 131        |
| 28.1.4 Execute data formatting programs – AEXCFMTPGM.....              | 132        |
| 28.2 Working with associated files for data recovery.....              | 132        |
| 28.2.1 Generate an associated file.....                                | 133        |
| 28.2.2 Execute recovery program with an associated file.....           | 133        |
| 28.2.3 Specify contents of standard routines for associated files..... | 134        |
| 28.3 Generating data formatting programs – AGENFMTPGM.....             | 135        |
| 28.4 Executing data formatting programs – AEXCFMTPGM.....              | 139        |
| 28.4.1 Set Example 1 – Data recovery.....                              | 142        |
| 28.4.2 Set Example 2 (with associated file) – Data recovery.....       | 143        |

## INTERNAL DESCRIPTIONS

---

|  |            |
|--|------------|
| <b>29 Internal descriptions</b> .....                                    | <b>146</b> |
| 29.1 Working with these cases in ARCAD Transformer Field.....            | 146        |
| 29.2 Using processes in internal descriptions.....                       | 147        |
| 29.3 Working with multi-format files.....                                | 149        |
| 29.4 Understanding file naming conventions in internal descriptions..... | 149        |
| 29.5 Locating applications in native IBM i-mode.....                     | 150        |
| 29.6 Locating applications in mode-36.....                               | 150        |
| 29.7 Referring to a file field by positional record.....                 | 152        |
| 29.8 Re-describing D.B. files.....                                       | 152        |

## NON-DATABASE FILES

---

|  |            |
|--|------------|
| <b>30 Working with non-database files</b> .....                        | <b>154</b> |
| 30.1 Understanding the particularity of multi-format files.....        | 154        |
| 30.2 Building lists of non-database files – PF36 field repository..... | 157        |
| 30.3 Propagating and modifying non-database files.....                 | 160        |
| 30.4 Generating recovery programs – non-database files.....            | 160        |



## OTHER FEATURES

---

|   |            |
|---|------------|
| <b>31 Simulating files in COBOL</b> .....                                   | <b>163</b> |
| 31.1 Internal description of files in the included source.....              | 163        |
| 31.2 Retrieving PF-INT fields list in COBOL – AFLDDCTINT.....               | 164        |
| 31.3 Simulating external descriptions in COBOL – AFLDDCTIN2.....            | 165        |
| 31.4 Extracting fields to process.....                                      | 165        |
| 31.5 Functioning of ACVTPGMFLD.....   | 165        |
| <b>32 Troubleshooting</b> .....   | <b>166</b> |
| 32.1 Modifying sources before propagation.....                              | 166        |
| 32.2 Considering the level of cross-references used for each component..... | 167        |
| 32.3 Propagating multiple applications.....                                 | 167        |
| 32.4 Avoiding incomplete operations in RPT, RPT38 or RPT36.....             | 167        |
| 32.5 Calling programs in SBMJOB by RTGDTA or by QCMDEXC.....                | 168        |
| 32.6 Transferring parameters by DS for multi-use.....                       | 169        |

# Tables

---

|  |     |
|--|-----|
| Table 1: Contact ARCAD.....  | 3   |
| Table 2: Related Documentation.....  | 4   |
| Table 3: ARCAD Transformer Field User Guide Publication Record.....                        | 4   |
| Table 4: Alphanumeric propagation 'C'.....   | 24  |
| Table 5: Alphanumeric propagation 'R'.....   | 25  |
| Table 6: Steps, Files, and Lists used in propagation.....                                  | 33  |
| Table 7: Summary of the main automatic modification functions.....                         | 49  |
| Table 8: Field length extension parameters.....  | 52  |
| Table 9: Parameters for screen modifications for an associated display field.....          | 53  |
| Table 10: Parameters for screen modifications where associated display field is blank..... | 54  |
| Table 11: Words indicating the nature of a source line.....                                | 56  |
| Table 12: Extraction macro required parameters.....  | 60  |
| Table 13: Extraction criteria.....   | 61  |
| Table 14: Filling the fields of an LSTF file.....  | 63  |
| Table 15: Options to configure Transformer.....  | 67  |
| Table 16: Options for propagation and modifications.....                                   | 67  |
| Table 17: Other Transform options.....   | 69  |
| Table 18: Details of propagation parameters.....   | 71  |
| Table 19: Modification parameters for DB file fields.....                                  | 75  |
| Table 20: Field addition parameters.....   | 76  |
| Table 21: Replacing file field name parameters.....  | 79  |
| Table 22: Screen/printer-level modifications parameters.....                               | 81  |
| Table 23: Adding another field parameters.....   | 84  |
| Table 24: Field format values.....   | 88  |
| Table 25: Standard routine languages.....  | 89  |
| Table 26: Options to implement routines.....   | 93  |
| Table 27: ACVTPGMFLD command parameters.....   | 101 |
| Table 28: ACVTDDSFLD command parameters.....   | 105 |
| Table 29: AWRKFRMCHG command parameters.....   | 108 |
| Table 30: Options for impacted sources.....  | 109 |
| Table 31: APRTFRMCHG command parameters.....   | 110 |
| Table 32: AAPYFRMCHG command parameters.....   | 123 |
| Table 33: Fields used to generate recovery programs.....                                   | 131 |
| Table 34: AGENFMTPGM command parameters.....   | 136 |
| Table 35: AEXCFMTPGM command parameters.....   | 140 |
| Table 36: Cases to work with ARCAD Transformer Field.....                                  | 146 |
| Table 37: Examples of positional reference.....  | 152 |

# Figures

---

|   |     |
|---|-----|
| Figure 1: ARCAD Transformer Field in the ARCAD product suite.....                 | 13  |
| Figure 2: Accessing ARCAD Transformer Field: Enter ARCAD on the command line..... | 19  |
| Figure 3: Accessing ARCAD Transformer Field: Select ARCAD Transformer.....        | 19  |
| Figure 4: Screen showing all TRANSFORM menu options.....                          | 20  |
| Figure 5: The propagation layout.....   | 32  |
| Figure 6: Newly-added fields.....   | 48  |
| Figure 7: Automatic modification of DSPFs/PRTFs.....                              | 51  |
| Figure 8: List of fields extraction macro.....                                    | 60  |
| Figure 9: Associated File Data Library.....                                       | 134 |



# INTRODUCTION

# 1 About ARCAD Transformer Field

*Transform field changes in evolving databases*

ARCAD Transformer Field is a suite of tools that automates the mass transformation of IBM i applications. Part of this suite, ARCAD Transformer Field is equipped to automate the mass transformation of source code when the structure of a database changes, such as when field sizes increase or types are changed. It updates your application automatically by validating the new lines of source code as proposed by the tool. All in all, it manages to save more than half of your development efforts via mass source code conversion while reducing the risk of error and safeguarding the reliability of your application.

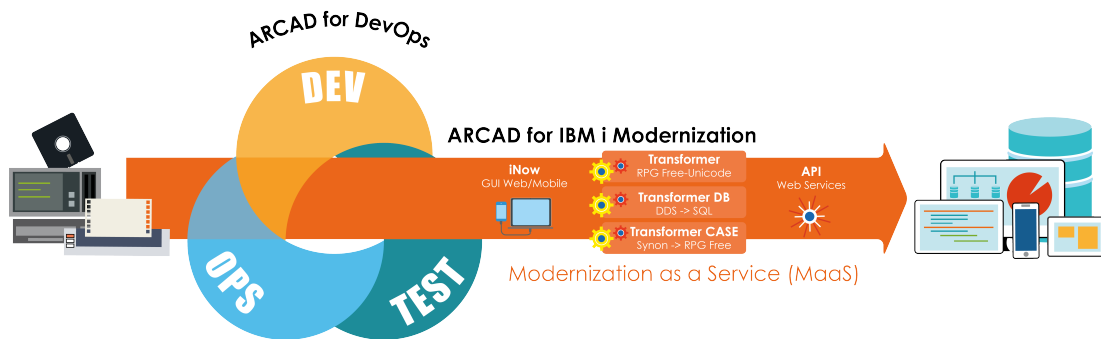


Figure 1: ARCAD Transformer Field in the ARCAD product suite

## 1.1 Functional constraints

There are a few functional constraints you may encounter when using ARCAD Transformer Field. These constraints are:

### 1.1.1 Understand cross-reference problems

ARCAD Transformer Field will not take components without cross-references into account. The propagation will fail and the entire process chain will stop if a cross-reference and its source are incoherent and there is no intercepted error message.

#### Example

Cross-references and sources may be incoherent if line numbers are offset, sources are modified and cross-references are not re-created, or if sources are transferred to production and cross-references are not re-created.

#### Important!

The ARCAD Repository and all cross-references must be up to date.

### 1.1.2 Work around incorrect files

To propagate, the previous file objects (\*FILE) (PF, LF, DSPF, PRTF) must be in the version. You may encounter problems if you have already applied the file modifications and compiled them.

If you restart the propagation or the automatic modifications (programs, DSPF/PRTF files), the new file will be taken into account (for example, to find the internal descriptions of fields and their position).

 **Important!**

Ensure that you do not compile new files in the version library too quickly.

### 1.1.3 Plan simultaneous processes

When planning such processes, a few pointers are to be kept in mind.

 **Warning!**

Do not run the transformer propagation engines and modifications simultaneously when the same components are being transferred to production (during the night, for example).

The transfer to production copies the sources first, and at the end of the process, it recreates the cross-references so there is a high risk of discrepancies.

 **Important!**

Transform only after updating the cross-references.

### 1.1.4 Flag or modify for multi-users

ARCAD Transformer Field is a multi-user product but you need to be careful.

In an open version, all flags and modifications concerning the same component must be run by the same user. It is the session user who runs the job that will allocate the modification propositions for the source line.

This may pose a problem if line modifications (for a specific program) were allocated to user 1 and others were allocated to user 2. This case often arises if the `ACVTPGMFLD` and `ACVTDDSF LD` commands which can allocate the same programs, are used by different users.

 **Important!**

If you have any doubts, verify that the flagged or modified components already exist for another user (with `AWRKFRMCHG`) before running the processes.

## 2 Pre-requisites

---

### Chapter Summary

|                                       |    |
|---------------------------------------|----|
| 2.1 Load the repository.....          | 15 |
| 2.2 Create a working environment..... | 15 |
| 2.3 Train for ARCAD.....              | 16 |

ARCAD Transformer Field requires that you have IBM i  $\geq$ v7.1.

 **Important!**

Due to a new licensing format, starting from v10.07.00 the ARCAD Transformer Field solution is only compatible with IBM i  $\geq$ v7.1. If you are running an older IBM i OS, you must upgrade.

### 2.1 Load the repository

---

For ARCAD Transformer Field to function correctly, you must first have completely configured the applications:

- Define the libraries linked to an application.
- Audit their contents.
- Resolve anomalies (duplicate sources, etc...).
- Load the component repository.
- Load the detailed field and component cross-references.

 **Important!**

- For ARCAD Transformer Field projects, the repository must be loaded with the following parameters:  
External fields only (RTVEXTFLD) \*NO  
Only used fields (RTVUSDFLD) \*NO
- ARCAD Transformer Field can only process files with sources. Use the `AGENFILSRC` command to generate any missing PF or LF sources.

### 2.2 Create a working environment

---

The options in the ARCAD Transformer Field menu need to be placed in an open version before being run.

In order to create this version, you need to:

- Declare a development environment if one isn't already declared (`ADCLENV`) (with, if necessary, \*NONE, \*NONE, ... in the environment's libraries).
- Open the version (`AOPNVER`).

**Note**

If you need to simply carry out an impact analysis without line markings or modifications, it is still necessary to open the version. You can remove it afterward (`ADLTVER`).

## 2.3 Train for ARCAD

---

In addition to the ARCAD Transformer Field concepts explained in this guide, it is necessary to follow the below basic training sessions:

- ARCAD Process Manager and ARCAD List Manager.
- ARCAD Repository and cross-references.
- Configuring an application.

**Reference**

For details about transferring to test or production with ARCAD Skipper, refer to the product's documentation.



## 3 Main concepts

---

### Chapter Summary

|   |    |
|---|----|
| 3.1 Detailed impact analysis for unmodified fields.....             | 17 |
| 3.2 Detailed impact analysis for automatically-modified fields..... | 17 |
| 3.3 Data recovery programs.....                                     | 18 |
| 3.4 Operational chronology for data recovery.....                   | 18 |
| 3.5 Quality over speed.....   | 18 |

After ensuring that the [Pre-requisites](#) are managed, the transformation process can begin.

### 3.1 Detailed impact analysis for unmodified fields

---

1. Build the list of fields to process serving as the platform for automatic propagation and modifications.  
[Working with field lists on page 59](#)
2. Configure the propagation.  
[Working with the TRANSFORM menu: Propagation and Transformations on page 67](#)
3. Run the propagation without modification (#1-2).
4. Analyze the quality of the propagation (list of anomalies, etc...).  
[Detecting anomalies generated by propagation on page 69](#)
5. If the quality is not satisfactory, do whatever is necessary to improve it and go back to #3.
6. Use the results.

### 3.2 Detailed impact analysis for automatically-modified fields

---

1. Repeat #1 to 5 above to ensure the quality of the propagation.
2. Configure the types of automatic modifications (extension or addition of file fields, automatic modifications in the programs, behavior for display/printer fields, ...).
3. Configure the file read/write, screen display/enter routines if necessary.
4. Delete the lines already marked in `AWRKFRMCHG`.
5. Run the modification of PF sources: `ACVTDBFFLD`.
6. Run the propagation with automatic program modifications: `ACVTPGMFLD`(#1-3).
7. Run the automatic modification for display/printer fields: `ACVTDDSFLD`.
8. Apply the modifications: `AAPYFRMCHG`.

If, at this stage, you see that the inserted routines are not correctly configured, it would be better to restart the whole process rather than making manual modifications in all the programs!

9. Consult, then compile the PF (and LF) modifications.

10. Consult, then compile the DSPF/PRTF modifications.
11. Consult, then compile the program modifications.
12. Test, correct, etc.
13. Transfer to test and/or production and distribute to sites.

**Note**

When extending a file field (or even if no action is carried out on file fields), you can avoid re-executing the entire propagation and only run #3 (having kept the marked lines).

**Reference**

It may be necessary to generate the recovery programs on the files in case the PFs were modified. For more information about this, refer to [Recovering data on page 129](#).

### 3.3 Data recovery programs

---

Data recovery programs automate a specific process which should be run for all the data contained in the file's fields, which are contained in the list of fields to be processed.

You can generate and run data recovery programs:

- just after building the list of fields to be processed (if you are absolutely sure it's complete),
- after propagating, with a detailed impact analysis, and completing the list of fields to be processed (following the analysis of the anomalies list), or
- after running and applying the automatic modifications, and after compiling the new PF in the version.

### 3.4 Operational chronology for data recovery

---

1. Configure the standard process routine to run for each field.
2. Generate these recovery programs (with compilation).
3. Execute the test files (in the version library).
4. Execute the actual files at specific dates in specific environments.

### 3.5 Quality over speed

---

ARCAD Transformer Field prioritizes the quality of the transformation results. It is important to verify the quality of the propagation results (with the anomaly lists) before using the transformed data. You run the risk of propagating incomplete or incorrect data if you exploit it immediately without analyzing the eventual anomalies detected.

## 4 Overview of ARCAD Transformer Field

Unlike ARCAD Transformer DB and ARCAD Transformer RPG, ARCAD Transformer Field is accessible via the native IBM i interface, popularly recognized as the green screen.

Accessing the options on the ARCAD Transformer Field interface is quite simple.

**Step 1** Log in to the native 5250 screen using your IBM i credentials. The following images will guide you further on reaching the TRANSFORM menu.

**Step 2** On the main menu screen, type **ARCAD** in the command line and press Enter.

```

MAIN                                IBM i Main Menu                                System:  CSPWTRN1
Select one of the following:
  1. User tasks
  2. Office tasks
  3. General system tasks
  4. Files, libraries, and folders
  5. Programming
  6. Communications
  7. Define or change the system
  8. Problem handling
  9. Display a menu
 10. Information Assistant options
 11. IBM i Access tasks
 90. Sign off

Selection or command
====> ARCAD
-----
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu
(C) COPYRIGHT IBM CORP. 1980, 2013.
  
```

Figure 2: Accessing ARCAD Transformer Field: Enter ARCAD on the command line

**Result** You will see the next screen with all the ARCAD modules.

**Step 3** Type **4** in the command line to access the transformer suite and press Enter.

```

ARCAD_ENG                            ARCAD - General menu (ARCAD)                    System:  CSPWTRN1
Select one of the following options:
  1. ARCAD-Audit          IBM i Audit & Restructuring          (ARAUDIT)
  2. ARCAD-Observer      Application analysis                  (OBSERVER)
  3. ARCAD-Integrator    Software reception                   (INTEGRATER)
  4. ARCAD-Transformer   IBM i refactoring tools              (TRANSFORM)
  5. ARCAD-Skipper       Software configuration management    (SKIPPER)
  6. ARCAD-Datachanger   Data configuration management        (ARDTC)
  7. ARCAD-Verifier       Regression testing                   (VERIFIER)
  8. ARCAD-Extract       Data extraction and scrambling       (EXTRACT)
  9. ARCAD-Deliver       Release Management                   (DELIVER)

Common options
 90. Sign off                SIGNOFF
 91. Macro-command menu     (ARMACCMD)
                               More...

Option or command
====> 4
-----
F3=Exit  F4=Prompt  F9=Retrieve  F10=Cmd entry  F12=Cancel  F13=Repos.
F14=List  F15=Macro  F20=Function  F21=Setup     F22=Main Menu
(C) Copyright 1992,2018 ARCAD Software
  
```

Figure 3: Accessing ARCAD Transformer Field: Select ARCAD Transformer

**Result** The next screen will display all of the options belonging to the transformer suite.

```

ARCAD_ENG  ARCAD - Impact Analysis/Source Convers Menu (TRANSFORM)
                                                    System:  CSPWTRN1

Select one of the following options:

  Current Environment
    1. Re-initialize the current environment      AINZCURENV
    2. Display current environment              ADSPCURENV

  Configure the propagation
    6. Work with propagation types              AWRKPRPTYP
    7. Work with standard routines             AWRKSTDRTN

  Manual impact analysis
    10. From the field repository               ADSPFLDDCT
    11. From a field or a component            ADSPFLDREF
    12. From the repository                    ADSPOBJREF

                                                    More...

Option or command
===> _____

F3=Exit   F4=Prompt  F9=Retrieve  F10=Cmd entry  F12=Cancel  F13=Repos.
F14=List  F15=Macro   F20=Function F21=Setup     F22=Main Menu

(C) Copyright 1992,2018 ARCAD Software

MW                                                    20/007
  
```

Figure 4: Screen showing all TRANSFORM menu options

You will be able to use all the options for transforming fields from this screen. These options are covered across various sections in the product documentation.



# PROPAGATION FUNCTIONS

## 5 Working with propagation functions

---

The propagation between 2 fields is based on the principle that each field character in a program (or even the entire field itself) is used only to store one type of information anywhere in the program. This information always has the same nature: either it is always the information you are looking for with the propagation or it is always something else.

The propagation analyzes the existing relationship between each analyzed field character and each character of the field to be propagated, according to the nature of the operation in the program's instructions and this for one or several lines, according to the language. When this principle is not verified, the propagation diverts itself towards those fields containing information that is not of the same nature. In this case, it may be necessary to set up propagation stoppers.

Propagation will only be carried out if:

- all the characters in an alphanumeric-type propagation are kept in the field to be propagated (beginning with the field format's upper-case letter, followed by all the lower-case letters).
- a minimum number of contiguous characters in a numerical-type propagation are kept in the field to be propagated (upper or lower-case characters of the field format).

The propagation is bi-directional, except in specific cases. According to the analyzed instruction, there is a flow of data from one field to another (but not vice-versa), or there is no data flow whatsoever (for instance, for a comparison instruction).

The propagation is run keeping the following instructions in perspective:

- Data allocating instructions.
- Test instructions (data comparison).
- Redefining field position instructions (data structure, mother fields, daughter fields).
- Character string position instructions (extraction of a part of the string, or concatenation of strings without deleting the blanks).
- Simple calculation instructions (addition, subtraction) (in the case of numerical propagation).
- More complex calculation instructions like multiplication or division in the case of numerical propagation and according to the behavior defined for this type of instruction.
- The instruction for defining one field in relation to the other (to be activated or deactivated in the propagation type, according to habitual programming methods).
- Parameter run instruction when calling a program (propagation in a field situated in the same row as the called program).
- Parameter reception instruction at the declaration of received parameters (propagation in the fields situated in the same row as the calling program(s)).
- Single key or key list instruction (when the field to propagate has a key).

## 6 Identifying fields

---

In order to identify the fields or parts of fields to be propagated, you must:

1. Build the list of fields to propagate.
2. Create a propagation type identified by a **letter**. The objective is both to define the behavior of the propagation and also the type of automatic modifications you want.
3. Allocate a field format containing this **letter**, to the fields in the list.

After ensuring the above, you can run the propagation.

- In step 1 (Entry in the programs), the propagation lists the different program fields corresponding to the fields to propagate while allocating their proper field format.
- In step 2 (Propagation), it analyzes all the instructions which use a propagated field, in order to determine whether other fields present in the same instruction are to be propagated. Later, these fields will be propagated themselves and so on.

 **Note**

The instruction analysis is not based on the logic design of the program (sequence of instructions). The instructions are analyzed separately from the rest of the program.

# 7 Outlining propagation behavior

## Chapter Summary


|  |    |
|--|----|
| 7.1 Alphanumeric-type propagation..... | 24 |
| 7.2 Numeric-type propagation.....      | 25 |

This section describes how to work with alphanumeric- and numeric-type propagation.

### 7.1 Alphanumeric-type propagation

**Rules:** The X characters to propagate in the original field to be propagated must be present in the new propagated field.

- There may be other characters in the field to be propagated.
- There may be other characters in the propagated field.
- The field to propagate must contain X characters to be propagated.
- The propagated field cannot be smaller than the number of characters to be propagated and must keep all the characters to be propagated.

 **Example**

The alphanumeric type of propagation ‘C’ must follow 5 characters (making up a code).

The instructions, as shown in the table below, use the operation codes RPG MOVEL (allocated on the left) and MOVE (allocated on the right).

A(5) = Alphanumeric of 5.

P(5) = Packed numeric on 5.

| Field to propagate and field format | Propagated field | Instruction       | Result field format and propagation | Y/N |
|-------------------------------------|------------------|-------------------|-------------------------------------|-----|
| RPCOD A(5) ‘C4c’                    | WPCOD A(5)       | MOVEL RPCOD WPCOD | ‘C4c’                               | Y   |
| WPCOD A(5) ‘C4c’                    | XPCOD A(5)       | MOVEL XPCOD WPCOD | ‘C4c’                               | Y   |
| RPCOD A(5) ‘C4c’                    | APZON A(8)       | MOVEL RPCOD APZON | ‘C4c3.’                             | Y   |
| RPCOD A(5) ‘C4c’                    | BPZON A(8)       | MOVE RPCOD BPZON  | ‘3.C4c’                             | Y   |
| RPCOD A(5) ‘C4c’                    | CPZON A(4)       | MOVEL RPCOD CPZON | ‘C3c’                               | N   |

Table 4: Alphanumeric propagation ‘C’



| Field to propagate and field format | Propagated field | Instruction          | Result field format and propagation | Y/N |
|-------------------------------------|------------------|----------------------|-------------------------------------|-----|
| RPCOD A(5) 'C4c'                    | DPZON P(5)       | MOVEL RPCOD<br>DPZON | 'C4c'                               | Y   |
| RPCOD A(5) 'C4c'                    | EPZON P(8)       | MOVE RPCOD<br>EPZON  | '...C4c'                            | Y   |

Table 4: Alphanumeric propagation 'C'

 **Important!**


Whether the propagated field is alphanumeric or numeric, the characters are propagated in a precise manner without reduction or enlargement of the number of propagated characters.

## 7.2 Numeric-type propagation

**Rules:** A minimum number of characters to propagate, present in the original field to be propagated, must be present in the new propagated field (but not always the first ones).

If the propagated field is declared as a numeric field, we can consider that the entire field is propagated.

- There may be other characters in the field to be propagated (if it is alphanumeric).
- All the characters in the field to propagate are to be propagated (if it is numeric).
- There may be other characters in the propagated field (if it is alphanumeric).
- All the characters in the propagated field are related to the letter (if it is numeric).
- The field to be propagated contains at least the minimum number of characters to be propagated.
- The propagated field contains at least the minimum number of characters to propagate.

 **Example**

The alphanumeric type of propagation 'R' must follow a numeric field with a length varying between the different fields (but having at least 3 characters).

The instructions, as shown in the table below, use the operation codes RPG MOVEL (allocated on the left) and MOVE (allocated on the right) and Z-ADD numeric allocation.

| Field to propagate and field format | Propagated field | Instruction          | Result field format and propagation | Y/N |
|-------------------------------------|------------------|----------------------|-------------------------------------|-----|
| RPVAL P(5) 'R4r'                    | WPVAL P(5)       | MOVEL RPVAL<br>WPVAL | 'R4r'                               | Y   |

Table 5: Alphanumeric propagation 'R'

| Field to propagate and field format | Propagated field | Instruction          | Result field format and propagation | Y/N |
|-------------------------------------|------------------|----------------------|-------------------------------------|-----|
| RPVAL P(5) 'R4r'                    | XPVAL P(7)       | Z-ADD RPVAL<br>XPVAL | 'R6r'                               | Y   |
| RPVAL P(5) 'R4r'                    | YPVAL P(4)       | Z-ADD YPVAL<br>RPVAL | 'R3r'                               | Y   |
| RPVAL P(5) 'R4r'                    | ZPVAL P(2)       | Z-ADD RPVAL<br>ZPVAL | 'Rr'                                | N   |
| RPVAL P(5) 'R4r'                    | APVAL A(7)       | MOVE RPVAL<br>APVAL  | '..R4r'                             | Y   |
| RPVAL P(5) 'R4r'                    | BPVAL A(4)       | MOVE RPVAL<br>BPVAL  | 'R3r'                               | Y   |
| APVAL A(7) '..R4r'                  | CPVAL A(10)      | MOVE APVAL<br>CPVAL  | '..R4r3.'                           | Y   |
| APVAL A(7) '..R4r'                  | DPVAL P(7)       | MOVE APVAL<br>DPVAL  | 'R6r'                               | Y   |

Table 5: Alphanumeric propagation 'R'

**⚠ Important!**

When the propagated field is numeric, it is considered as propagated on all of its characters (digits in this case).

## 8 Languages processed by the propagation

---

Analysis of all the instructions working on the variables (allocation, test, declaration, parameter change, calculus, string process).

The following languages are analyzed for the propagation:

- RPG (RPG38) and SQLRPG
- CBL (CBL38) and SQLCBL
- CBLLE (with specification in column 6) and SQLCBLLE
- Classic RPGLE and SQLRPGLE
- Free RPGLE without specification, but using column 8-80
- Fully-free RPGLE
- CLP or CLLE
- The commands created in an application (CMD, CMD38)RPG36 (propagation similar to RPG in the instructions, but with a different entry point)
- OCL36 (Sorting cards, ?x? and LDA parameters, BLDFILE, BLDINDEX orders)

The analysis of the instructions working directly on the files are:

- SQL instructions situated in the sources SQLRPG, SQLRPGLE, SQLCBL, SQLCBLLE.
- Instructions OPNQRYF and CPYF situated in the sources CLP or CLLE.
- Instructions CHGVAR in CLP or CLLE, when they are destined to build the character string placed in selection variables of an OPNQRYF.

### Languages not processed by the propagation

- The language C.
- Other languages for which the ARCAD suite doesn't have detailed field cross-references like (FORTRAN, BASIC, etc...).

**Note**

The ILE procedures and commands will be integrated in the propagation process once they have detailed field cross-references.

# 9 Evaluating lists of impacted fields


## Chapter Summary

|  |    |
|--|----|
| 9.1 Evaluating lists of physical file fields.....                  | 28 |
| 9.2 Evaluating lists of program fields.....                        | 28 |
| 9.3 Evaluating lists of display or printer fields (DSPF/PRTF)..... | 29 |
| 9.4 Evaluating lists of fields in the L.D.A. to propagate.....     | 29 |

This list you created contains the list of fields for which you want to analyze the links.

## 9.1 Evaluating lists of physical file fields

This is the most common case when using ARCAD Transformer Field.

 **Reference**  
 For more information about the creation of this list, refer to [Working with field lists on page 59](#).

This list only takes the physical file fields into account (PF, PF38). There is no need to include any logical file fields (they will be deducted by the propagation).

It can sometimes be useful to include (or exclude) the reference file fields or physical file fields. This only has an impact in the case of propagation with automatic modification of the file sources.

## 9.2 Evaluating lists of program fields

[*Optional*] It may be necessary to analyze the impact on one or more program fields that we know of, in order to establish the list of the other related program fields, as well as those of the file fields.

Follow the subsequent steps to establish the list of related program fields.

**Step 1** Use the `ACRTFLRLST` command to create a list of all the fields of one or more programs.

```
Field cross-references (ACRTFLRLST)

Type choices, press Enter.

By field, literal or component      *FLD          *FLD, *OBJ, *IFS,
*LIT
Field name . . . . .
+ for more values
Application ID . . . . .          *CURENV       Character value,
*CURENV
Component display level . . . . . *CURENV       Character value,
*CURENV
Only if the field is used . . .    *USED         *USED, *IMPL, *UPD...
TO list . . . . .                LSTTMP        Name
```

```

Library . . . . . QTEMP Name, *LIBL, *CURLIB,
*CURENV
Replace or add record . . . . *REPLACE *REPLACE, *ADD
Retrieve system function tree . *NO *YES, *NO
1st selection field . . . . . *FLDUSE Character value,
*BLANK
2nd selection field . . . . . *FLDNAME
Type of list to generate . . . . *LST *LST, M, O, F

```

**Step 2** Use the `AEDTLST` command on the created list (with Option 3=Copy) or `AEXTLST` (extract the items from a list) for moving the field or fields into the list of fields to process.

**Warning!**  
 You could have also created an item in the list with `AADDLSTE`, but in this case you would have to correctly specify each required argument.

**Note**  
 In COBOL, several fields can be homonyms when they are described in groups of different fields. In ARCAD, the name of these fields is followed by 'DUPxx' in order to get the names of the unique fields. You must verify using the repository which field you want to choose.

### 9.3 Evaluating lists of display or printer fields (DSPF/PRTF)

You may also want to begin an impact analysis on one or more display or printer fields to determine with which program or file fields they are linked to.

In this case, use the `ACRTEFLRLST` command to create the list of display (or printer) fields.

### 9.4 Evaluating lists of fields in the L.D.A. to propagate

If the fields to be propagated are systematically present in most of the L.D.A. part (common to the entire application), you can add them to the list of fields to propagate (using F6) as follows:

Add an entry to the list (AADDLSTE)

Type choices, press Enter.

```

Object/member/field . . . . . Name, *IFS
Library (object or source) . . *LIBL Name, *CURLIB, *LIBL, *IFS
Type (object/source/field) . . . Character value, *ONLY
List . . . . . *CURRENT Name, *CURRENT
Library . . . . . Name, *LIBL, *CURLIB
Check existence of element . . . *YES *YES, *NO
IFS object . . . . .
Destination directory . . . . . *DFT

```

Additional Parameters

|                                    |         |                            |
|------------------------------------|---------|----------------------------|
| Type of list . . . . .             | *LST    | *LST M O F I               |
| Object attribute/component typ     | *OBJMBR | Character value, *OBJMBR   |
| Compil type/Field format . . . .   | *OBJMBR | *OBJMBR, *BLANK, Type      |
| (F4=List                           |         |                            |
| Source file/field file . . . . .   | *OBJMBR | Name, *OBJMBR, *BLANK      |
| Creation library . . . . .         | *OBJMBR | Name, *OBJMBR, *BLANK      |
| Object/member modif date . . . .   | *OBJMBR | Date, *OBJMBR, *BLANK      |
| Object/member change time . . . .  | *OBJMBR | Time, *OBJMBR, *BLANK      |
| # for compilation sequence . . . . | *OBJMBR | Number, *OBJMBR            |
| Flag . . . . .                     | ' '     | X, ' ', A, H, L            |
| 1st selection field . . . . .      | *BLANK  | Character value, *BLANK... |
| System function tree info:         |         |                            |
| Sub-system . . . . .               | *OBJMBR | Character value, *OBJMBR   |
| Function . . . . .                 |         | Character value            |
| Sub-function . . . . .             |         | Character value            |
| Text . . . . .                     | *OBJMBR |                            |

The name of the **P0017L0009** field indicates the position and length concerned.

The Type A(9) shows the type and length and the format **Z** specifies the field format.

The attribute must be \*LDA.

The format and file field are at **X** because a valid character must be entered.

Therefore, each time the LDA is used in RPG(LE), CLP, or OCL36, any eventual variables loaded from this LDA position will be propagated. When extending fields, the length and position will be extended to take into account the size increase of this field (even if the position in question wasn't defined in the program).

# 10 The propagation layout

---

## Chapter Summary

|   |    |
|---|----|
| 10.1 Propagating lists of fields – LSTxxxx                  | 33 |
| 10.2 Propagating lists of programs to be processed – LSTPGM | 34 |
| 10.3 Propagating lists of propagation stoppers – LSTPRPSTP  | 34 |
| 10.4 Propagating lists of D.D.S. fields – LSTDDSFLD         | 34 |
| 10.5 Propagating lists of derived fields – LSTDRVFLD        | 36 |
| 10.6 Propagating source line files – AARFCHSF1              | 37 |
| 10.7 Propagating other AARF... files                        | 38 |
| 10.8 Propagating source lines                               | 39 |
| 10.9 Propagating lists of other D.B. fields                 | 40 |
| 10.10 Propagating lists of anomalies                        | 41 |

The following diagram presents the layout of the lists and files used by the propagation:

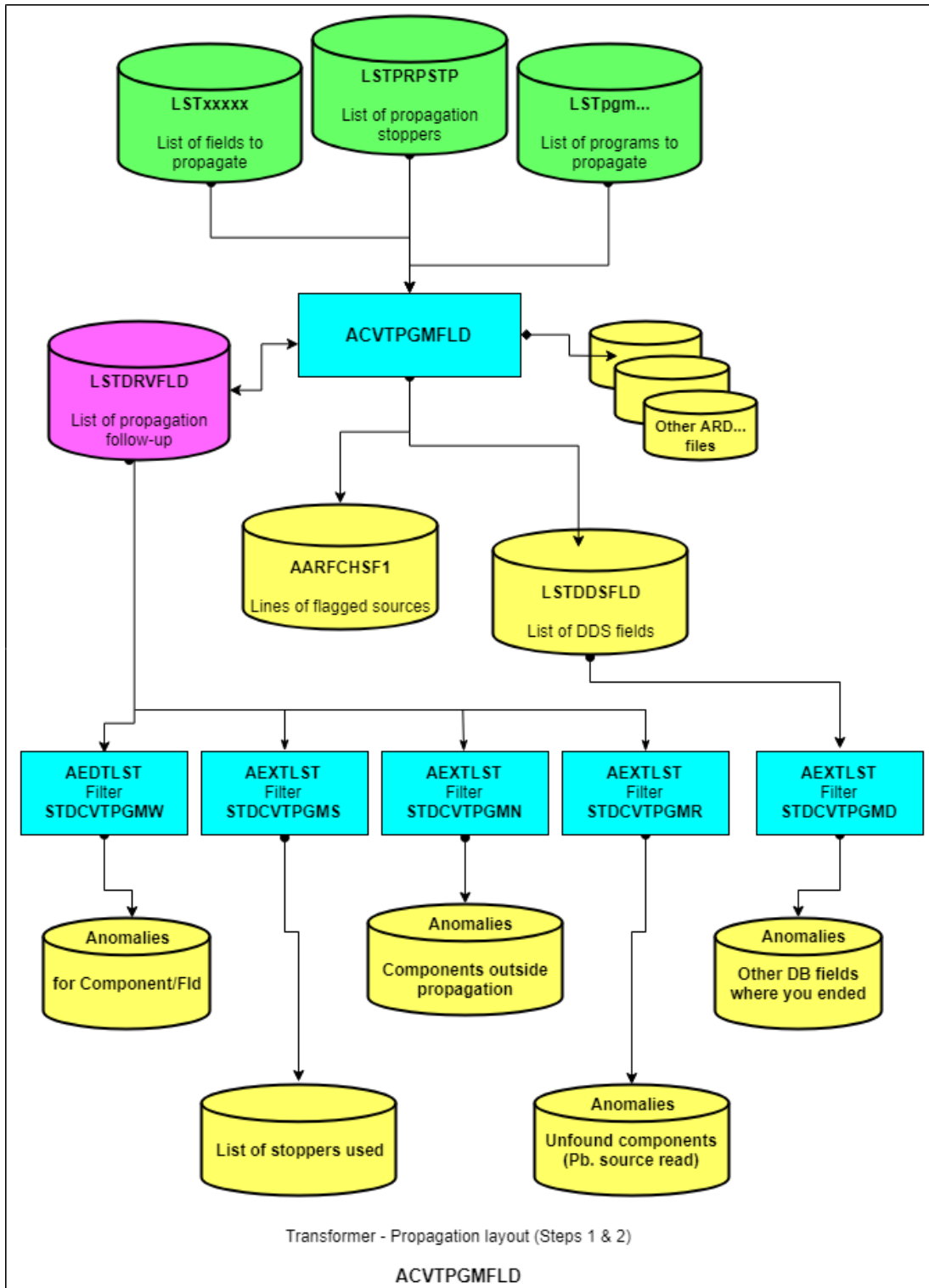


Figure 5: The propagation layout

This layout does not include:



- all the files of the ARCAD database, repository, cross-reference, etc.
- the sources of all the application components used.

**Note**

The names of the lists indicated here are used as a default; they can be modified at the start of propagation.

The best place to store these lists is the \*CURENV version library (except for the starting LSTxxxx list of fields to be propagated).

The following table shows the steps with the files or lists used:

| Step 1 (Entry in the programs)  | Step 2 (Propagation)   |
|---|--|
| <b>Input:</b> <ul style="list-style-type: none"> <li>• LSTxxxx: List of Fields to propagate</li> <li>• and possibly LSTPGM - List of Programs to be processed</li> </ul>    | <b>Input:</b> <ul style="list-style-type: none"> <li>• LSTPRPSTP: List of stoppers</li> <li>• and possibly LSTPGM ... List of Programs to be processed</li> </ul>              |
| <b>Input/Output:</b> <ul style="list-style-type: none"> <li>• LSTDRVFLD: List of derived fields</li> </ul>  |  |
| <b>Output:</b> <ul style="list-style-type: none"> <li>• LSTDDSFLD: List of D.D.S. fields</li> <li>• LSTDRVFLD: List of derived fields</li> <li>• Other ARD files</li> </ul> | <b>Output:</b> <ul style="list-style-type: none"> <li>• LSTDDSFLD: List of D.D.S. fields</li> <li>• AARFCHSF1: Flagged source lines file</li> <li>• Other ARD files</li> </ul> |

Table 6: Steps, Files, and Lists used in propagation

When step 2 is finished (or even running), you may reference the following anomaly lists:

- Other D.B. fields where the propagation ended
- Component/Field anomalies
- List of stoppers used
- Components outside propagation
- Components not found (problem reading source)

## 10.1 Propagating lists of fields – LSTxxxx

Another possible denomination is the list of impacted fields.

This list is the starting point for the propagation and the modification engines (except the ACVTDDSFLD).

This is a list of fields to be processed (file or program or display/printer field) and is often the result of the 1st level impact.

## 10.2 Propagating lists of programs to be processed – LSTPGM

---

This list is not indispensable for the propagation; by default, all the programs in the application can be processed.

If you wish to carry out the propagation for one program, it is not necessary to create a list; simply indicate the name of the program to process when starting the propagation.

However, to define a specific domain for the propagation on a set of programs you must use this list (after choosing the name); this list must be a list of source members (type M) created by one of the following:

- an `ACRTMBRLST`
- an `ACRTXRFLST`
- an extraction from another list of source members.

When run with this list i.e. LSTPGM, the propagation will be limited to programs in this list only, possibly with the COPY clause sources included.

Each parameter entry between a program on this list and one not on the list will fill in the list of fields of components outside propagation.

In the case of propagation with internal descriptions, do not forget to include the calling CLP (when these substitute the file names) or the OCL36 in this list.

## 10.3 Propagating lists of propagation stoppers – LSTPRPSTP

---

This list contains the propagation stoppers that you defined with `ASETPRPSTP`.

In step 2, it is used at the beginning of a program field process to verify if this field is part of an \*EXCLUDE stopper. If it is at YES, the field will not be processed.

In step 2, the propagation to a field for which you enabled a \*VIAFLD type stopper triggers the search for a field, loading or using this field or the associated field (propagation to a field with what may be described as having a “leap-frog” behavior).

The \*LDA type stoppers (LDA re-organization) are used in the 3 stages of the propagation (Steps 1, 2, and 3).



### Reference

For more information about Step 3 (Automatic modifications in the programs), refer to [Internal descriptions on page 146](#)

## 10.4 Propagating lists of D.D.S. fields – LSTDDSFLD

---

This list can also be called the list of DB and DSPF/PRTF fields to be impacted.

This output list from the propagation contains all the external references of a field propagated to a file (physical, logical, display, printer).

It is built by the propagation in the following way:

### *In Step 1 (Entry in the programs):*

- Physical file fields in the starting list of fields to propagate.
- Logical file fields that depend on these physical files.

These are stored as follows:

- `LST_JOB` File field name
- `LST_JLIB` File format name
- `LST_JSRCF` File name (physical or logical)
- `LST_CATR` File attribute (PF, PF38, LF, LF38)
- `LST_CTYPE` Field type
- `LST_CTXT` Extended field format
- `LST_JZSEL2` Blank

In addition, if you have the internal descriptions of files (in RPG, COBOL) you will also find the following in this list:

- All the references (program + code line n°) that correspond to the definition of the file field in the program (stored this time by the position and length in the record, and not by the field name).
- `LST_JOB` Positional reference to file field
- `LST_JLIB` File format name
- `LST_JSRCF` File name (physical or logical)
- `LST_CATR` File attribute (PF, PF38, LF, LF38)
- `LST_JZSEL2` Program name + : + line n° + : + abbreviated program type
- `LST_CTYPE` Field type
- `LST_CTXT` Extended field format

If a file field is described under several field names in a program, there will be those many items in this list (one for each line).

### *In Step 2 (Propagation):*

New fields from physical and logical files found by the propagation (these are the other database fields in which the propagation ended).

- Field for the Printer/Display files found by the propagation
- `LST_JOB` File field name
- `LST_JLIB` File format name
- `LST_JSRCF` File name (physical or logical, display, print)
- `LST_CATR` File attribute (PF, PF38, LF, LF38, DSPF, DSPF38, PRTF, PRTF38)
- `LST_CTYPE` Field type
- `LST_CTXT` Extended field format
- `LST_JZSEL2` If an unexpected new PF/LF field: Program name and program line where it came from.

In addition, if you have the internal descriptions (in RPG, COBOL) of new file fields, the name of the field added to the list is: Positional reference to file field.

If this is a DSPF, DSPF38, DSPF36, PF, PF38, PF36, LF, LF38, LF36 field, the file name is placed in `LST_JSRCF` (as this external file does exist).

However, if it is fields in card O (RPG) or on an Output Buffer (COBOL) for a Print for which no external PRTF... file exists, the information is stored as follows:

- `LST_JOB` Positional reference to file field (OxxxxLyyyy)
- `LST_JLIB` Program name
- `LST_JSRCF` Program name
- `LST_CATR` Fictive attribute PRTF36
- `LST_JZSEL2` Program name + ':' + line n° + ':' + Abbreviated program type
- `LST_CTYPE` Field type
- `LST_CTXT` Extended field format

## 10.5 Propagating lists of derived fields – LSTDRVFLD

---

This is also known as the list of fields impacted in the programs or the list of propagation follow up.

This propagation output list includes all the fields of the programs concerned by the propagation.

It contains 3 records at the beginning of the file, whose objective is to visualize the progress of the ACVTPGMFLD processes in Steps 1, 2, and 3.

It is built by the propagation in the following way:

*In Step 1 (Entry in the programs):*

- Name in the program for the physical or logical file's field.
- FILE.FIELD name for the fields of the files used in the SQL queries in the SQLRPG(LE), SQLCBL(LE) program.
- FILE.FIELD name for the fields of the files used in OPNQRYP or CPYF in CLP.
- `LST_JOB` Program field name if its name has a maximum of 10 characters.
- Fictive name of the field CB00000001, CB00000002 in the opposite case.
- `LST_JZSEL2` Program field name
- `LST_JLIB` Program name
- `LST_JSRCF` Program name
- `LST_CATR` Program source type (RPG, RPGLE, CBL, CLP, etc...)
- `LST_CTYPE` Field type
- `LST_CCPLT` Field format code
- `LST_CTXT` Text with: file name, format and field causing the propagation
- `LST_CSTS` ''

If getting the program field is obtained from the internal descriptions of files, the information is the same except for the text:

- `LST_CTXT "Fld:xxxxx-Fmt:xxxxxx Fii: xxxxx,PF(xxxxx)"`

Fld = The file field in question

Fmt = The file format

Fii = External name of file, its type and its internal name in the program.



#### Reference

For more information, refer to [Interpreting the nature of a source line on page 56](#).

*In Step 2 (Propagation):*

- The propagation process of a program field will create new items in this list for each new propagated field. The only differences in the contents are:
  - `LST_CTXT` - Part of a source line that triggered the propagation (in which the original field can be seen).

However, the `LST_CSTS` status will have the following values:

- [blank]: non-processed field
- X: processed field
- S: field not processed as it is being subject to a propagation stopper
- N: field outside propagation
- R: field whose process triggers an access error to a source
- B: DB field present in a non-propagated file
- C: previously-enlarged field
- D: ambiguous field
- E: DB field present both in a propagated file and another non-propagated file
- F: DB field present both in a propagated file and in a DSPF or PTRF!
- G: DB field present both in a propagated file and another non-propagated file, and also in a DSPF or PTRF.

When the process is active in *Step 2*, the percentage displayed corresponds to the number of items on this list that were already processed. You can see them by typing `DSPPFM LSTDRVFLD`.

Column 81 corresponds to `LST_CSTS` (not blank, if processed); the records are processed in the order of their entries to this list, as seen by the `DSPPFM`.

## 10.6 Propagating source line files – AARFCHSF1

This ARCAD-based file contains the flagged source lines first, then those modified by the propagation and the automatic modifications. Consider that it contains the source modification propositions.

Its primary key is the following:

- Application code
- Development environment
- Source lines destination version
- Source member name
- Source member type
- Line N°
- Sub-line N°

**Note**

This is the only file that is not necessarily purged when it needs to start a propagation. This is why it can be purged as follows:

- For the version in question, use the command `AWRKFRMCHG` and `F17=Delete all`.
- Or possibly a `CLRPFM` of this `AARFCHSF1` file **only** if you are sure you are the only person using ARCAD Transformer Field and also, that you have the authorizations required for such an operation!

**Important!**

As it refers to the source lines of the principal component, it is important that the reference source is not modified between the propagation and the automatic modifications that fill this file, and applying it towards a new source in the version. If not, the modifications will be off (in relation to the source line) and result in complete disorder!

## 10.7 Propagating other AARF... files

The propagation also updates other files in the ARCAD database. There is no need to go into any details of these files.

**Important!**

- These files as a principal key, each has the Library and the Name of the list of derived fields (normally, the version library + `LSTDRVFLD`).
- When you begin a propagation step, all the records concerning the list of derived fields processed, are deleted beforehand.

However, they keep the information of the last propagation; that is why, when you no longer need this information, it will be necessary to purge these files through `ACLDRVFLD` - Delete the info. generated by the propagation.

## 10.8 Propagating source lines

---

In the LSTDRVFLD list and by the display screen for derived fields (ADSPDRVFLD), you can find an explanation for the reason of the propagation for each propagated field (in LST\_CTXT).

This information can contain:

### 1. Items resulting from *Step 1: Entry in the programs*

- \* Propagation by the external descriptions of the file field to the file field in the program.
  - File:xxxxxx,xxx Fld:xxxxxx Fmt:xxxxxx
  - File = Name of file in question, file type
  - Fld = The file field in question
  - Fmt = The file format
- \* Propagation by the external descriptions of the file field to a program field used as a key.
  - KFLD-->File:xxxxx,xxx Fld:xxxxxx or
  - KLIST-->File:xxxxx,xxx Fld:xxxxxx
- \* Propagation by the internal descriptions of the file field to the file field in the program.
  - Fld: xxxxx Fmt:xxxxxx Fii: xxxxx,PFxx(xxxxx)
  - Fld = The file field in question
  - Fmt = The file format
  - Fii = External File name, its type and its internal name in the program.
- \* Propagation by the internal descriptions for a key field.
  - Key:xxxxx Fmt:xxxxxx Fii: xxxxx,PFxx(xxxxx)
  - Key = The key field in question
  - Fmt = The file format
  - Fii = External File name, its type and its internal name in the program.
- \* Propagation from a program field indicated in the starting list.
  - Direct-->program, field

### 2. Items resulting from *Step 2: Propagation – Source line*

In most cases, an extract of the source line in the program that caused the propagation can be seen here. The line is “compressed and then possibly truncated.” Also, the name of the field in question is present in this line.

In the case of parameter transfer, you also have:

- \* The propagation of a parameter to the called program:
  - Calling pgm: call instruction or
  - Calling pgm-->called ILE pgm: call instruction if the call is from the principal module of a pgm.
- \* The propagation of a parameter to a calling program:

- Called pgm: call instruction or
  - Called pgm<-calling ILE pgm: call instruction if the call was from the principal module of a pgm.
3. Items resulting from *Step 2: Propagation – Special cases*
- \* Propagation between an internal D.S. and another external part (in RPG...)
    - External D.S.: Field
  - \* Propagation between the external fields and the internal fields for the file buffers (in COBOL)
    - External Record Fmt: Field
  - \* Propagation between the Input and Output buffers in a COBOL program for which the display descriptions are internal:
    - Field->I/O DSPF

## 10.9 Propagating lists of other D.B. fields

---

This list of **anomalies** does not exist as a list file on the drive:

- It is a visualization of the LSTDDSFLD by the `AEDTLST` command with the filter `STDCVTPGMD`.
- Set of fields in external reference to a data file that was detected by the propagation in Step 2.

This is the **essential** anomaly list for refining the quality of the propagation by allowing you to detect the following cases:

- a. oversights in the original list of fields to propagate.
- b. interface file fields that you don't want propagated but which are linked to certain propagated fields.
- c. additional unnecessary fields placed in the list of fields to be propagated (which then detect other DB fields).
- d. propagation quality problems which need to be refined.
- e. field cross-references not up to date in relation to the sources.

A high-quality propagation should only contain the **interface** file fields.

### How to analyze the fields in this list

In order to determine which case this is, you must:

1. find the name of the field in this list (LST\_JOB) and the name of the program (and the program line in the case of an internal description),
2. go in and read the derived fields in order to visualize the path followed by the propagation, and
3. consult the source(s) concerned.



## What to do next:

1. Oversight of field in the original list. For a field oversight, add it to the list of fields to propagate.



### Reference

For more information, refer to the [Recovering data on page 129](#)

You might also need to take a look back at the extraction macro.

2. Detection of an interface file field. It is normal that they are present in this list. This can be processed by `ACVTDDSFELD`.
3. One field appears too many times in the list of fields to propagate. Simply remove them from the list of fields to propagate.
4. Propagation quality problems often resulting from programming techniques that differ from ARCAD Transformer Field principles.
  - A field must always contain information of the same nature. In this case, you can either use stoppers, or adapt the sources in question.
5. Cross-reference update problems.

## 10.10 Propagating lists of anomalies

---

This section presents the anomalies that do not exist as a list file on the hard disk.

### 10.10.1 Propagate lists of anomaly component fields

---

This list of **anomalies** doesn't exist as a list file on the hard disk:

- It is a visualization of the `LSTDRVFLD` using the command `AEDTLST` with the `STDCVTPGMW` filter.
- A set of program fields for which an anomaly was detected.

#### Note

Do not process this list before having dealt with all the anomalies seen in the [Propagating lists of other D.B. fields on the previous page](#).

Each field of a file presented in error in the list is also displayed in this list for one or more programs.

Besides the program fields in file anomaly, the following also appear in this list:

- A program field with an ambiguous format (that is, where the propagation has detected that a position in the field contains both the propagated characters (identified by a letter) and other characters (identified by another letter).
- Previously enlarged fields (if you have asked for their detection when you ran `ACVTPGMFLD`).

### 10.10.2 Propagate lists of stoppers used

This list of **anomalies** doesn't exist as a list file on the hard disk:

- It is a visualization of the LSTDRVFLD using the command `AEDTLST` with the `STDCVTPGMS` filter.
- A set of program fields for which the process was not carried out as it corresponds to one of the stoppers set up.

This allows you to verify that you correctly used the defined stoppers (and which ones).

### 10.10.3 Propagate lists of components outside propagation

This list of **anomalies** doesn't exist as a list file on the hard disk:

- It is a visualization of the LSTDRVFLD using the command `AEDTLST` with the `STDCVTPGMN` filter.
- A set of program fields that cannot or should not be processed. These can be:
  - a field as an input parameter of a program present in an application, but which won't be processed as you defined a propagation stopper to an application or a specific list of programs to be processed (however, this program is called by one of the processed programs).
  - a field situated in a program, present in an application, but which won't be processed as you have defined a propagation limited to one application or a specific list of programs to be processed; everything outside this field will be transferred as parameters when calling a processed program.
  - the call indication by a processed program from a program that doesn't have any cross-reference fields.

In this case, the type of program indicated is `*PGM` and the field name is the name of the calling program.

The analysis of this list allows you to verify the impacts outside an application, or outside the list of components to be processed. This can lead you to:

- look back over the list of components to be processed,
- load the overlooked component cross-references,
- or above all, highlight the links with the components outside the application.

### 10.10.4 Propagate lists of components in source access error

This list of **anomalies** doesn't exist as a list file on the hard disk. It is a visualization of the LSTDRVFLD using the command `AEDTLST` with the `STDCVTPGMR` filter.

A set of program fields for which the process is impossible as ARCAD Transformer Field cannot access a component's source. This is often:

- a source line included in the principal source by a `COPY` clause. These sources without object are not part of the application repository.
- an error during the loading of `RPG/RPGLE` cross-reference for a `COPY` clause source line on which the access to a file by key, is found. If there is confusion between the `COPY` clause and the file, then in this case, recreate the component cross-references (this anomaly has since been corrected from ARCAD version `V_6.05.F` onwards).

- a file defined in the keyword FORMAT (file) by the OPNQRYF order and which doesn't exist as an object on the hard disk.

# 11 Understanding field formats

---

## Chapter Summary

|  |    |
|--|----|
| 11.1 Indicating field formats in lists of propagated fields..... | 44 |
| 11.2 Field format codes.....                                     | 45 |
| 11.3 Compressed field formats.....                               | 46 |
| 11.4 Extended field formats.....                                 | 46 |

The field format allows you to define the exact structure of a field for ARCAD Transformer Field. It contains information on the nature of the field (propagation type), the length of the field, and the positions concerned within the field.

There are 3 visual levels that exist for the field format:

- field format code
- compressed field format
- extended field format

The 3 field formats are a different representation of the same information.

## 11.1 Indicating field formats in lists of propagated fields

---

This field format is indicated by the user in the original list of fields to be propagated. To simplify data entry, this field format has an automatic length for the letter corresponding to a propagation type.

- In an alphanumeric type propagation, the uppercase letters A-Z automatically correspond to the number of characters to be propagated.



### Example

C will give you a C5c field format if the number of characters to propagate for the letter C is 6 characters.

- In a numerical type propagation by indicating the uppercase letters A-Z, the propagated field format letter will adapt to the length of each field.



### Example

T gives you the format T12t for a field with 13 characters and T5t for a 6-character field.

### 11.1.1 Field format contents

---

The following characters can be found in a compressed or extended field format:

- The dot (.) shows a character that is not concerned by the propagation.
- The uppercase letters (A-Z) note the beginning of the propagated characters in the field.

- The lowercase characters (a-z) are usually situated after the equivalent uppercase letter. They complete the propagated characters.
- The set of 1 uppercase letter + x lowercase letter(s) gives you the full set of propagated characters.
- The numbers from 0 to 9 are only used in a compressed field format to specify the occurrence of the non-number character that follows.
- The parentheses ( and ) mark the position of the beginning and end of a packed field's characters contained in an alphanumeric field.
- The square brackets [ and ] mark the position of the beginning and end of a binary field's characters contained in an alphanumeric field.
- The dash (-) contained between parentheses or brackets corresponds to the middle characters in a packed or binary field contained in an alphanumeric field.
- The slash / is used as a separator between the end of the field and the format, then clarified for each binary or packed field contained in an alphanumeric field.

#### Note

The characters from A-Z are only significant if they correspond to a letter defined as a type of propagation. If this is not the case, they can be used for representing other characters (to be compared, in this case, to a dot (.)).

#### Example

Here are some examples of field format:

- **A**: A field to be propagated on a character.
- **Aaaa** or **A3a**: An A type field to be propagated on 4 characters.
- **...Aaa..**: An 8-character field in which the propagated part corresponds to the 3 characters situated in position 4, 5, and 6.
- **Aaa..Aaa..**: A 10-character field containing two A type fields, with 3 characters to propagate, in position 1 and 6.
- **104.A9a86.**: A 200-character field containing a field with 10 characters to propagate, in position 105 to 114.
- **(--)/A6a/**: An alphanumeric 4-character field **(--)** in which these 4 characters contain a field packed on 7 digits in an **A6a** format.
- **13.(50.(5-)28./Ttt/T12t/**: An alphanumeric field with 100 characters with certain positions containing type T packed fields; at position 14, on 2 characters **(50)**, packed field of the type **Ttt** (3 digits); at position 66 on 7 characters **(5-)**, packed field of the type **T12t** (13 digits).

## 11.2 Field format codes

The field format code is the only identifier for defining the format of a field. It is coded with a maximum of 7 characters, as follows:

- If it is entered by the user and allocated to a field on the propagation entry list, it is recovered in its present status. (Examples: **T**, **..C**, etc...)

- If it is generated by the propagation and the compressed field format has a maximum of 6 characters followed by a \$. (Examples: **T14t \$**, **.C5c \$**, etc....)
- If it is generated by the propagation and the compressed field format has more than 6 characters; this means it is a code containing a \$ followed by a 6-digit number. (Example: **\$000005**, etc....)

## 11.3 Compressed field formats

---

The compressed field format corresponds to the exact image of the field's format, but where each consecutive occurrence of at least 3 times for the same character is replaced by the occurrence value for 1, 2, or 3 digits followed by the character itself.



### Example

The compressed field format **..C4c8.C4c15.**

Corresponds to an extended field format **..Ccccc.....Ccccc.....**

It is this type of field format that is taken from the derived fields display screen.

## 11.4 Extended field formats

---

The extended field format corresponds to the exact image of the field format, character by character. It contains a maximum of 2000 characters. Each character situated beyond this limit will be completely ignored by the propagation.



### Example

**..Ccccc..** corresponds to a field of 9 characters in which the positions 3 to 7 contain 5 characters followed by the propagation.

# 12 Working with automatic modification functions

## Chapter Summary

|  |    |
|--|----|
| 12.1 Modifying logical files.....                                      | 48 |
| 12.2 Modifying program fields automatically.....                       | 49 |
| 12.3 Modifying display/printer fields automatically – ACVTDDSF LD..... | 51 |
| 12.4 Adding a file field.....  | 53 |
| 12.5 Modifying interface file fields automatically.....                | 55 |

The following 3 types of modifications are possible for the fields of a file:

1. **E - Extension of field length:** You indicate the number of extension (or reduction) characters to be implemented for each impacted field.

By default, the length modification of a file field will lead to the same modification being made to all the derived program fields.

2. **A - Addition of a file field (or even 2 fields):** This is the addition of a new field called a 2nd field (just before, just after, or at the end of a record) for each impacted file field. This new field can have the same characteristics (Type/Length) as the reference field, or you can create a particular type and length for the field (Field \$\$\$F2).

The name for the new fields will possibly be proposed before the actual addition which is carried out automatically; you should check this and if necessary, correct the new field names.

You can include a dependent link at file key level (physical or logical).

You can include the duplication of program code lines that use the propagated field, in order to obtain a similar line to the new file field (line duplication by imitation for an associated field).

It is also possible to simultaneously add another file field, called a 3rd field. (Field \$\$\$F3).

3. **D - Deletion of a file field:** The goal of this type of modification is to eradicate the fields which have become obsolete in the files and following that, remove the program code lines related to them.

For any type of modification chosen (A, E, D) you can include the insertion of routines in the programs (placed just after reading the file, or before writing the file).

### Important!


There can be an absence or presence of reference file fields in the impacted file fields.

Normally, the reference file fields are never used in a program; they are there only to allow the descriptions of file fields by external reference to fields in reference files.

- If you place the reference file fields in the list of fields to be processed:
  - The description of reference file fields will be modified (field extension or addition).
  - The PF fields referring to a reference file field will not be modified in the case of field extension

(there will just be a flag put in).

- In the case of field addition, these will refer to a new reference file field.
- If you leave out the reference file fields from the list of fields to be processed:
  - The reference file will not be modified.
  - The PF fields referring to a reference file field will be *relatively modified* in the case of field extension.

 **Example**  
 Before FIELDFI R REFFLD  
 After FIELDFI R +3 REFFLD

- The newly-added fields will be defined directly in the source of the PF.

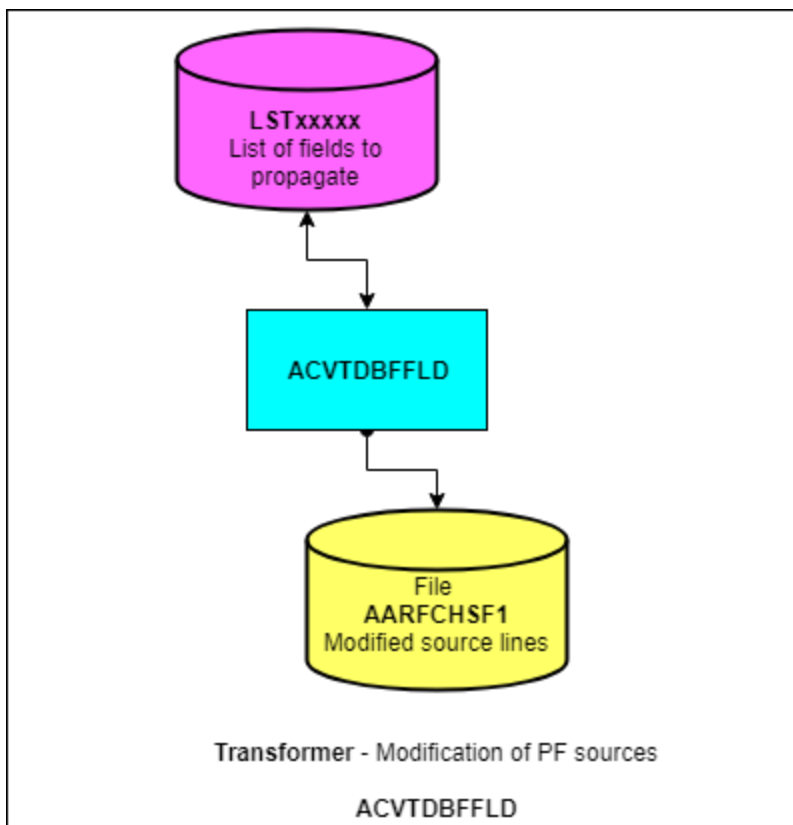


Figure 6: Newly-added fields

## 12.1 Modifying logical files

Don't be surprised, but these files sometimes need to be modified too (directly re-declared fields, newly-added fields, added keys).



The `ACVTPGMFLD` (Step 3) carries out this modification and not the `ACVTDBFFLD` which is only used with the physical files. The reason for this is that it is the `ACVTPGMFLD` (Step 1) which propagates the physical file fields to the logical file fields.

## 12.2 Modifying program fields automatically

In the table below you will find a summary of the main automatic modifications which can be carried out on the programs by using ARCAD Transformer Field.

| Modifications  | Summary  |
|--|--|
| Automatic modification of programs - <code>ACVTPGMFLD</code> | In the case of a field extension, this process: <ul style="list-style-type: none"> <li>• extends all the propagated program fields (either in the default way, if the field has an external description in relation to the file or by modifying the field definition length.</li> <li>• it recalculates the field positions (in the case of internal file descriptions) so that they correlate with the files.</li> <li>• inserts routines, if necessary.</li> </ul> |

*Table 7: Summary of the main automatic modification functions*


| Modifications  | Summary  |
|--|--|
| <p>Process before writing and after reading D.B. files</p> | <p>In the case of field deletion, this process:</p> <ul style="list-style-type: none"> <li>deletes the lines of codes containing the entirely propagated fields.</li> <li>shortens the fields containing the deleted fields among other things.</li> </ul> <p>However, in this case a manual verification of modified sources is <b>obligatory</b> to ensure the correct logic of line deletions, notably in the case of a test instruction deletion (the ENDIF is not deleted).</p>   |
|  | <p>In the case of internal file descriptions, the process recalculates the field positions in order for them to be in correlation with the files.</p>  |
|  | <p>In the case of field addition, this process:</p> <ul style="list-style-type: none"> <li>duplicates by imitating the field process lines (either just the allocations or all the code lines).</li> <li>If requested in the configuration (similar function for 2 fields), it duplicates by imitating the field process lines (either just the <b>allocations</b> or all the code lines).</li> <li>The imitation used for naming the new program files tries to analyze the logical file that was adapted for the field names between: <ul style="list-style-type: none"> <li>The original field (z11) which already has a new associated field name (z12).</li> <li>The original field (z11) and the field to which it is propagated (z21).</li> </ul> </li> </ul> <p>The synthesis of these 2 logical rules can allow you to correctly name the field associated with the new propagated field (z22).</p> |
|  | <div style="border: 1px dashed green; padding: 5px;"> <p> <b>Example</b></p> <p>File: OFNBH (z11) OFNBJ (z12)</p> <p>Program: W1NBH (z21) W1NBJ (z22)</p> <p>Code lines: MOVE W1NBH TO OFNBH</p> <p>Added line: MOVE W1NBJ TO OFNBJ</p> </div>  |

Table 7: Summary of the main automatic modification functions

It should be noted that the imitation doesn't function in every case. The new field is, therefore, named with a number (9, 8, 7, 99, 98, 97, etc...) overwriting the last characters. For clarity in your program, you will need to rename these new fields using more explicit names.

- If a dependent link is defined for the new field, it is added (in RPG, RPGLE) to the list of keys. If necessary, a KLIST is added: a key from a single item (previous field) is replaced by a new KLIST containing the previous field and the new field.
- In the case of internal descriptions for files, it recalculates the field positions in order for them to be in correlation with the files. It also declares the new fields in the programs (when the previous field is present).
- If necessary, it can insert the routines:

- Process before writing, after reading the D.B. files (active on the original field, new field and 3rd file field).
- Process before writing, after reading the D.B. files to separate or switch the original field and the new field to one whole field (if you requested the file field name to be replaced in the programs).

## 12.3 Modifying display/printer fields automatically – ACVTDDSFLD

The programs undergoing a second modification process for each field linked to the fields processed by the `ACVTDDSFLD` command (Display/Printer fields):

- Add a routine before Display / after Input.
- But sometimes need to rename the display field in the program.
- Or also need to declare the Display/Printer fields directly in the program (notably for the fields in O card printout without PRTF, or for the internal descriptions of the display and printer fields).

The propagation in the `LSTDDSFLD` list results gives all the display or printer fields that are linked to the propagated fields (and also the PF/LF fields).

This list is the base of the display and printer field modification process.

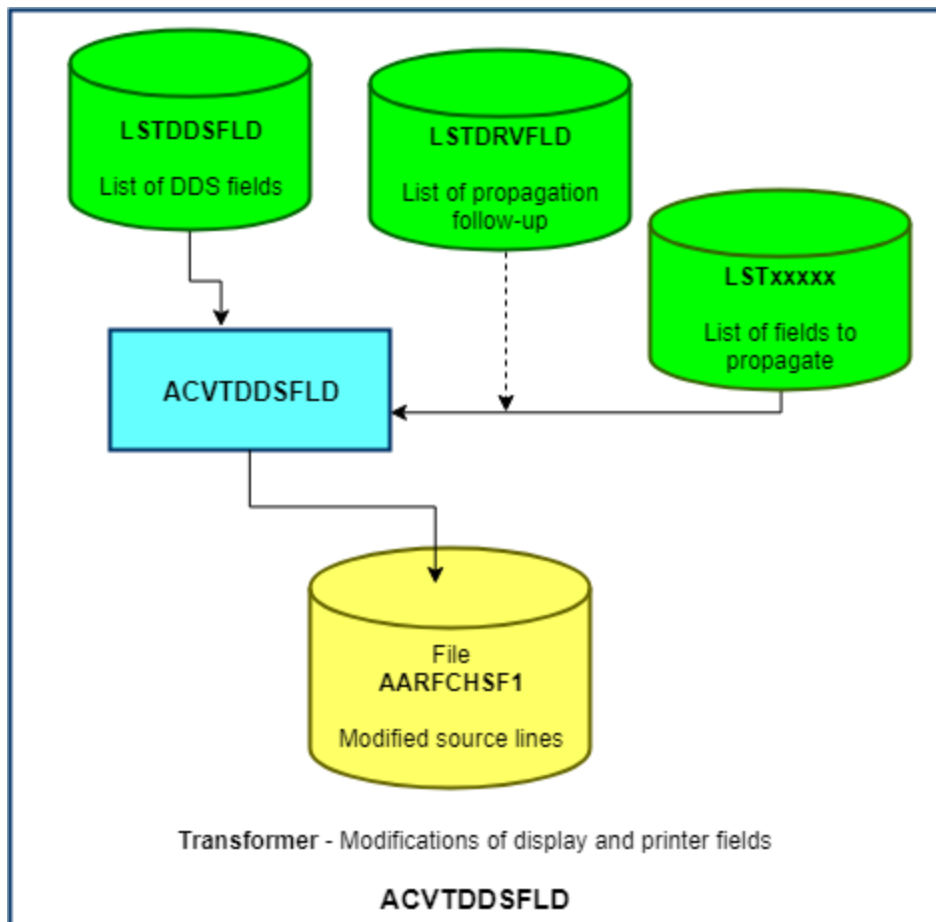



Figure 7: Automatic modification of DSPFs/PRTFs

It is possible to request the following modifications for the display fields (according to the modification at file level).

### 12.3.1 Field length extension

 **Example**  
 3-character extension of quantity fields (shown here on ZQT1 4,0):

```
Before:      ! Qty : ZQT1  Unit Pr : xxxx,xx !
!_____!
```

The table below presents the field length extension parameters:

| Parameters                 | Description  |
|----------------------------|--|
| *ALWAYS:<br>Always visible | <p>The display/printer fields visible by the user will directly mirror the field's length extension. If more space is needed, the line on which the field is present will be slightly re-organized (deletion of blanks and shortening of text).</p> <p>If even more space is needed, the display field is enlarged anyway; a "warning" is placed on the DDS of the display/printer field indicating a field overlap. No new display field is created (neither visible, nor hidden).</p> <pre>After:      ! Qty ZQT1___ Unit Pr : xxxx,xx ! !_____!</pre>   |
| *NEVER:<br>Never visible   | <p>You do not wish the user to be affected by the field extension. However, the display field was considered as enlarged in the program; it is, therefore, necessary to carry out specific processes to ensure the correct functioning of the display (or printer) file:</p> <ul style="list-style-type: none"> <li>◦ The field visible by the user will be renamed but will stay at the same position on the screen. (Field \$\$\$F2)</li> <li>◦ The field having the previous name will no longer be visible (either declared on the screen in a hidden field or declared in the program). It will have the new length (Field \$\$\$F1).</li> <li>◦ It will be necessary to insert processes to change the \$\$\$F1 values to \$\$\$F2 before the screen display or from \$\$\$F2 to \$\$\$F1 after input; you will define these by using one or two display/printer routines.</li> </ul> <pre>After: ! Qty : XQT1 Unit Pr : xxxx,xx ! ZQT1 7,0 hidden field !_____\$\$\$F2_____! \$\$\$F1</pre> |
| *IFFITS: If it fits        | <p>*IFFITS corresponds to *ALWAYS as long as you can extend the field, otherwise this becomes *NEVER when there is not enough room in the line.</p>  |

Table 8: Field length extension parameters

## 12.3.2 Field deletion

\*ALWAYS: **Always visible** is automatically established if the field is still present on a screen.

## 12.4 Adding a file field

Two different cases must be distinguished, and also the addition of a 3rd display field:

### 12.4.1 Associated display fields (\$\$A1)

You have requested a behavior similar to the newly-added file field and the functioning by imitation has allowed you to reach right up to screen level; the display field (\$\$F1) therefore already has an associated field name for the new field in the program (\$\$A1).

#### Example

Propagated unitary price; the addition of a file field of the same length.

ZPXU P(6,2) propagated

And associated field ZPAU loaded in the program

```
Before:      ! Unit Pr : ZPXU  !
! _____ !
```

The table below presents the screen modification parameters.

| Parameter   | Description   |
|---|---|
| *ALWAYS: Screen modification always visible           | <p>The newly-added field is placed in the screen, just before or just after the propagated field.</p> <p>If there is not enough room, the field will be placed as a hidden field.</p> <pre>After:      ! Unit Pr : ZPXU  ZPAU  ! ! _____ \$\$F1  \$\$A1  _____ !</pre>  |
| *NEVER: Screen modification not visible for the user. | <p>A new field is managed, but the screen must not be modified visually; the value displayed will either be that of the previous field (\$\$F1), or of the newly associated field (\$\$A1). To avoid any risk of program error, the display field will have a new name (\$\$F2).</p> <p>It is by the intermediary of a routine inserted before display or after reading, that either the previous field value (\$\$F1) or the new field's value (\$\$A1) will be sent to the screen in the \$\$F2 field.</p> <pre>After: ! Unit Pr : XPXU_____ ! Hidden fields ZPXU(\$\$F1) ! _____ \$\$F2 _____ ! and ZPAU. (\$\$A1)</pre> |

Table 9: Parameters for screen modifications for an associated display field

### 12.4.2 No associated display fields (\$\$A1 is blank)

You have not requested a behavior similar to the newly-added file field, or the functioning by imitation did not allow you to reach the screen; the display field (\$\$F1), therefore, has no associated field name in the program for the new field (\$\$A1 is blank).

In this case, you should only begin a screen modification if you can simulate the value of the new field (for example, with a conversion before the display carried out by a routine).

 **Example**

A propagated unitary price, addition of a file field with the same length.

ZPXU P(6,2) propagated and no associated field

```
Before:  ! Unit Pr : ZPXU      !
! _____ !
```

The table below presents the screen modification parameters:

| Parameters   | Description  |
|--|--|
| *ALWAYS:<br>Screen modification "Always visible".        | <p>The newly-added field is placed in the screen, just before or just after the propagated field.</p> <p>If there is not enough room, the field will be placed as a hidden field.</p> <p>Its name will be generated at the last moment.</p> <pre>After:  ! Unit Pr : ZPXU  XPXU ! ! _____ \$\$F1  \$\$F2  !</pre>  |
| *NEVER:<br>Screen modification not visible for the user. | <p>No new field is managed, and the screen must not visually be modified; the value displayed will either be that of the previous field (\$\$F1) or the value calculated from \$\$F1 simulating the value of a new field. To avoid any risk of program error, the display field will have a new name (\$\$F2).</p> <p>It is by the intermediary of a routine inserted before display or after reading that it will either be the value of the previous field (\$\$F1) or a value calculated from \$\$F1 which will be sent to the screen in the \$\$F2 field.</p> <pre>After:  ! Unit Pr : XPXU      ! Hidden field ZPXU ! _____ \$\$F2      !</pre> |

Table 10: Parameters for screen modifications where associated display field is blank

### 12.4.3 Add a 3rd display/status field (\$\$F3)

The objective of this 3rd display field is particular; it has nothing in common with a possible 3rd field added to the PF files.

This is a field for which the value should be allocated by one way or another just before the display or tested just after the input (using one or two standard routines).

Whether you previously put \*NEVER or \*ALWAYS will have no impact on this addition.

If requested, this 3rd field will be systematically added (either visibly if there is enough room, or as a hidden field if not).

### Example

In the previous case, add a 3-character Currency Code field.

With an associated field, at \*ALWAYS:

```

                CPXU
After:      ! Unit Pr : ZPXU   ZPAU   Eur !
! _____ $$F1 _____ $$A1 _____ $$F3!
```

With an associated field, at \*NEVER:

```

                CPXU
After: ! UnitPr:XPXU Eur! Hidden fields ZPXU ($$F1)
! _____ $$F2 _____ $$F3 _____ ! and ZPAU ($$A1)
```

Without an associated field, at \*ALWAYS:

```

                CPXU
After:      ! Unit Pr : ZPXU   XPXU   Eur !
! _____ $$F1 _____ $$F2 _____ $$F3!
```

Without an associated field, at \*NEVER:

```

                CPXU
After: ! Unit Pr : XPXU   Eur ! Hidden field ZPXU
! _____ $$F2 _____ $$F3 _____ !
```

## 12.5 Modifying interface file fields automatically

The propagation results give you the LSTDDSFLD list in which the display or printer fields linked to the propagated fields (and also the PF/LF fields) are present.

In this list, a certain number of PF-LF fields are considered anomalies:

List of other database fields where the propagation ended.

This list can sometimes serve as a base for the automatic insertion of a process (via ACVTDDSFLD).

However, it is necessary that this list only contains the interface file fields, that is, the file fields that you do not wish to modify (no field addition or extension) but for which you wish to simulate after each reading of (or before each writing of) the field extension or addition carried out previously in other files.

This is also only possible if you possess the different ways to define a functionally suitable conversion process through a standard routine.

With these routines, we can also insert a voluntarily erroneous conversion process, but which indicates to the programmer that this program should be adapted.

## 13 Interpreting the nature of a source line

Following the propagation and automatic transformation stages, the source lines (placed in the ARDCHSF1 file and accessible using `AWRKFRMCHG`) include a word indicating the nature of the marking and the modification. The following table displays the list of different possible words:

| Category   | Words      | Description   |
|--|------------|---|
| Marked source lines                                    | MRK...     | A non-modified propagated line.   |
|  | MRKDEF     | A non-modified line containing the definition of a propagated field.  |
| Modified source lines                                  | CHGDEF     | A changed field definition.   |
|  | CHGPOS     | A changed field position (in I Card, O Card or in a Data structure) in the case where there is a position shift due to earlier extensions or field additions. |
|  | RENFLD     | A renamed field.  |
|  | CHGSST     | A position or length changed in a SUBSTRING (RPG,RPGLE,CLP).  |
|  | CHGOVL     | An overlay position changed in an overlay(xxx :y) in RPGLE.   |
|  | CHGKEY     | A changed key (in a KLIST).   |
|  | LENPOS     | A modified length and position in a DSPF/PRTF.  |
|  | MSGLEN     | A shortened message size in a DSPF/PRTF.  |
|  | CHGDDS     | A modified DDS DSPF/PRTF line.  |
|  | CHGEDT     | A modified Edit Code.   |
|  | CHGFIL     | A modified file declarative (internal description).   |
| Deleted source lines                                   | DLTDEF     | Deleted field definition.   |
|  | DLTFLD     | Deleted field.  |
| Added source lines                                     | NEWDEF-I   | A new definition of a newly-added field.  |
|  | NEWFLD-I   | A duplicated instruction for processing the newly-added field (and the associated fields).  |
|  | RENFLD-I   | A renamed field.  |
|  | HIDDEN-I   | A new field definition, added as a hidden field.  |
|  | NEWKEY-I   | A new field added as a key.   |
| Added source lines originating from a standard routine | INS*BEFORE | A source line inserted <b>before</b> file update, write, or display.  |
|  | INS*AFTER  | A source line inserted <b>after</b> file update, read, or input.  |
|  | INS*ONCxxx | A source line added to a program <b>once</b> .  |

Table 11: Words indicating the nature of a source line

When applying the modifications, the following character will be placed in position 5 of the source line:



- V: Verified line (Only marked)
- C: Changed line (Modified)
- I: Inserted line (New line)
- D: Deleted line (if still present in comment)



# IMPACT ANALYSIS

# 14 Working with field lists

---

## Chapter Summary

|   |    |
|---|----|
| 14.1 Creating/refreshing lists of database fields – FLDDCT..... | 59 |
| 14.2 Building field lists.....                                  | 59 |
| 14.3 Setting the field format – ASETFLDFMT.....                 | 63 |

Lists of fields to transform serve as a base for field conversion. To convert fields you must:

- (Re-)generate the list of all database fields.
- Build the list of fields to be processed.
- Define a field format for each of these fields.

## 14.1 Creating/refreshing lists of database fields – FLDDCT

---

Even when the database repository is correctly loaded (visible through `ADSPFLDDCT`), it is necessary to generate it in the form of a list. Use option **Create/Refresh list of database fields** of the TRANSFORM menu which runs the `FLDDCT` macro. The macro will:

- initialize the application environment (put the correct libraries online).
- create an empty LSTDBFFLD list in the library which should contain the application lists.
- build the list of the application database files (\*FILE, type PF or PF38 or TABLE) (in the LSTPF list).
- use this list (for each physical file) to build the LSTDBFFLD list which will then contain all the fields for all the physical files of the application.

### **Important!**

The resulting list is not maintained; it is necessary to restart the FLDDCT once files are added (or modified) in the repository.

You can then visualize this list with the following option **Display the database fields list**.

## 14.2 Building field lists

---

All the processes in ARCAD Transformer Field require a list of fields to process (fields to be propagated) as a starting point.

### **Reference**

For more information about the contents of this list, see [The propagation layout on page 31](#).

You can build this list (for file fields) several different ways. Whichever method is used, you will have the same results each time.

### 14.2.1 Create a field extraction macro-command

This EXT... macro-command always has the same role: it starts with the list of database fields (LSTDBFFLD) to obtain the list of fields to propagate (LSTxxxx), by successive extractions.

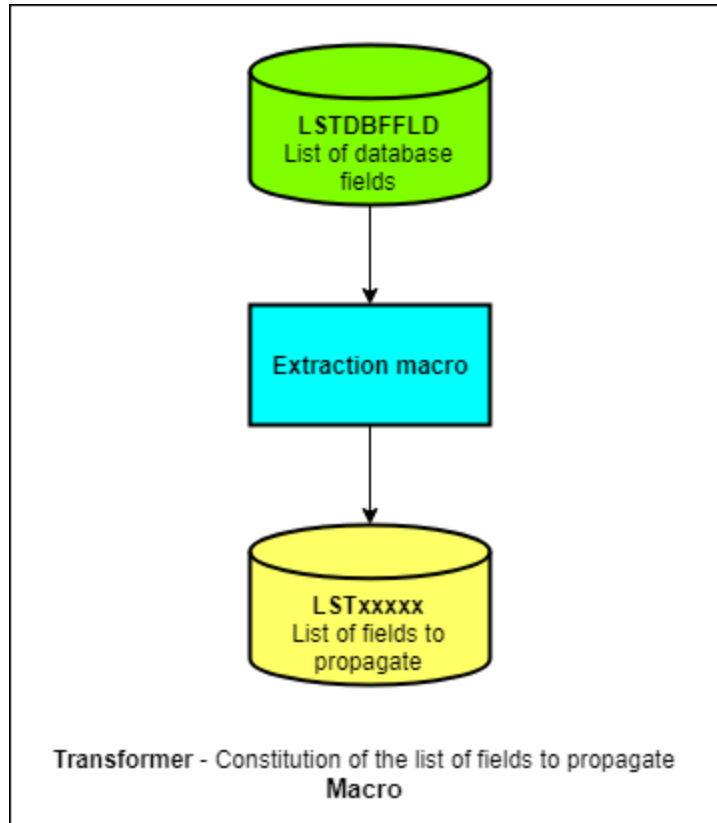


Figure 8: List of fields extraction macro

| Parameter        | Description                              |
|------------------|--|
| Application ID   | (as it is an *APP type macro)            |
| Entry list       | (&PFL) normally contains the LSTDBFFLD   |
| Destination list | (&PTL) enter the name of the result list |

Table 12: Extraction macro required parameters

The resulting list is created by default in the (&ALSTLIB) library; this is the library where your application lists are stored.

The macro must:

- create the result list LSTxxxx.
- create, when necessary, the intermediate lists in QTEMP.
- extract the fields in the LSTDBFFLD list (or in the intermediate list) using `AEXTLST` and place them in the LSTxxxx list (or an intermediate list).

- repeat as much extraction as necessary.
- delete the Field/FMT/File duplicates from the resulting list (using an `AUPDLST`).
- and if necessary, display or print the resulting list.

The easiest way to create your macro-command is to copy one of the standard macro-commands (with `AWRKMCCMDEXTFLD` and `3 Copy to XXX/EXT...`), then to modify it using `AWRKMCCCMD XXX/EXT...`

Preferably, create your macro-command in a personal library or in a library of your application.

 **Reference**

For more information about macro-commands and extraction command, refer to the Help in the ARCAD Process Manager and ARCAD List Manager menus.

Base the extraction on one or more of the following criteria:

| Criteria   | Description   |
|--|---|
| Library/File format<br>( <code>LST_JLIB</code> )             | Contains the name of the file format.   |
| Object/member/field<br>( <code>LSOBJ</code> )                | Contains the field name.  |
| Type<br>(object/source/field)<br>( <code>LST_CTYPE</code> )  | Contains the type and length of the field. <b>A(lg)</b> if the field is alphanumeric, <b>S(lg,nb dec)</b> or <b>P(lg,nb dec)</b> if the field is an extended or packed numeric field. |
| Object/type component attribute<br>( <code>LST_CATR</code> ) | Contains the type of PF: PF or PF38   |
| Text (contains type)<br>( <code>LST_CTXT</code> )            | The field's text.   |
| Source file/field file<br>( <code>LST_JSRCF</code> )         | Name of physical file.  |

Table 13: Extraction criteria

To run the macro, start the command (either directly or via a `SBMJOB`) then press `F4` to define the parameters.

#### 14.2.1.1 Review the list of fields

Use the `AEDTLST` command and define the name of the result list (with its library, if necessary) and press `F4`. The options list `LST_CSTST > FIELDS`.

This allows you to obtain the options adapted to a list of file fields in the screen that follows.

The field list displayed may not be the one you expected. It may contain excess fields (not concerned by your problem) or it may not be able to contain fields (even if you think they are useful) if they do not meet the selected criteria.

In this case, try to improve your macro (with other extraction criteria). Continue repeating this process until you get satisfying results.

 **Warning!**

Be careful. Restarting the extraction macro will completely re-execute every action. Therefore, you must wait until it is finished before removing (or suspending) any fields.

If only a few fields are missing, because they do not meet the selected criteria, you can easily retrieve them from the list of database fields and copy them manually to the list of fields to propagate. `AEDTLST LSTDBFFLD` displays all the database fields. Enter 3 next to the field(s) chosen and enter **TOLIST (lib/LSTxxxx)** at the end of the screen.

In case of any doubts, and to avoid adding the same field several times, finish by entering `AUPDLST lib/LSTxxxx` which will remove any eventual duplicates.

## 14.2.2 Build the list of fields to propagate from ARCAD Observer

If ARCAD Observer is accessible on a PC connected to your system, you can navigate in the field repository (database dictionary) and build the list of fields to process by using filters.

Then in ARCAD Observer, request an output of your selection in the form of an ARCAD list.

 **Reference**

For more information about this subject, refer to the ARCAD Observer documentation.

## 14.2.3 Retrieve the list of fields to process

You may also already have the list of fields to process on your IBM i or in a PC file.

In this case, using SQL or a program you wrote, you can populate the list directly.

 **Important!**

You must always have created the destination list beforehand, using the `ACRTLST` command.

You must be able to completely fill in the necessary fields of the list file (ARCAD lists are all in a LSTF file format).

Write a CLP that reads your file and adds the fields to the list of fields to process by copying from the LSTDBFFLD:

- `DUPLSTE LST_JOB(`File/Field name Format)
- `LST_CTYPE`(field type A(5), etc..)
- `FROMLIST`(LSTDBFFLD)
- `TOLIST`(Lib/LSTxxxx)
- `CTLELEM`(\*NO)
- `LST_JSRCF`(File Name)

You can also write a program that will create entries in the LSTF list file (with an OVRDBF before execution to the destination list).

For a list of fields of a file to propagate, the fields of the LSTF file are to be filled in as given in the following table:

| Field Name | Type | Length | Title                      | Contents for list of fields  |
|------------|------|--------|----------------------------|------------------------------|
| LST_CTYPE  | A    | 1      | TYPE OF ITEM (M/O/F)       | F for list of Fields         |
| LST_JOB    | A    | 10     | OBJECT/MEMBER/FIELD        | Field Name                   |
| LST_JLIB   | A    | 10     | LIBRARY/FIELD FORMAT       | File Format Name             |
| LST_CCPLT  | A    | 7      | COMPILE TYPE OR PROPAG FMT | Field Format                 |
| LST_CTYPE  | A    | 10     | OBJECT/MEMBER/FIELD TYPE   | Field Type: A(5) P(6,2)      |
| LST_CATR   | A    | 10     | OBJECT ATTRIBUTE           | PF or PF38 File Type         |
| LST_JSRCF  | A    | 10     | SOURCE FILE/FIELD FILE     | Physical file name           |
| LST_JXLIB  | A    | 10     | CREATION LIBRARY           | Name of file's library       |
| LST_TDATE  | A    | 10     | DATE                       | Blank                        |
| LST_NORD   | P    | 2      | ORDER N°                   | 0 (zero)                     |
| LST_CSTS   | A    | 1      | STATUS                     | Blank                        |
| LST_CTXT   | A    | 50     | TEXT                       | Text if available            |
| LST_TTIME  | A    | 6      | TIME (HHMMSS FMT)          | Blank                        |
| LST_JZSEL1 | A    | 10     | SELECTION FIELD            | Blankw                       |
| LST_JZSEL2 | A    | 30     | SELECTION FIELD 2          | Blank (or name of new fld 3) |
| LST_CSST   | A    | 12     | TREE                       | Blank (or name of new fld)   |

Table 14: Filling the fields of an LSTF file


## 14.3 Setting the field format – ASETFLDFMT

This operation is absolutely necessary as it is by this bias that the link between the fields and the types of propagation is carried out. The goal is to allocate a field format for each field.

Normally, the field format should be placed in the list in the LST\_CCPLT (field format) field.

However, even though the width of this field is limited to 7 characters, it is possible in particular cases to indicate a compressed field format of 50 characters by using the text field (LST\_CTXT). Its case:

- enter TEXT in the field format (LST\_CCPLT),
- enter the compressed field format in LST\_CTXT (Text)

 **Example**  
(This is the most complicated case possible) - Format of a file field "FILLER"  
of 100 characters (type of propagation A):

- Contains a field to propagate on 9 digits in position 21 to 29 (extended),
- and another field to propagate on 15 digits in position 62 to 69 (packed on 8 characters).

Field format (LST\_CCPLT)-> TEXT

Text-> 20.A8a32.(6-)31./A14a/

Several methods are possible to set the format for each field:

### 14.3.1 Set the same field format for all fields

Use the ASETFLDFMT command as follows:

```
Set field format (ASETFLDFMT)
```

Type choices, press Enter.

```
Application ID . . . . . *CURENV      Character value, *CURENV
List of fields to be modified > LSTSLFLD Name
Library . . . . . > *LIBL          Name, *LIBL, *CURLIB, *CURENV
Database file library . . . . > *NONE      *LIBL, *FROMLIST, *NONE
```

Additional Parameters

```
Default format:
Field length . . . . . > *OTHER      Number, *OTHER
Format . . . . . > A                Character value, *NONE
+ for more values
Restart after abnormal end? . *NO      *YES, *NO
```

Replace XXXXXXXX with the letter(s) constituting the field format.

**Note**

The ASETFLDFMT command contains other possibilities, notably to determine the field format according to the data contained in the files or to the length of the fields.

### 14.3.2 Set a particular format for specific fields

**Step 1** Use the AEDTLST command, specifying the name of the result list (with its library when possible) and choosing the options list `LST_CSTST > FIELDS (F4)`.

**Step 2** Select option 2=Change and press F4.

**Step 3** Define the field format in **Compile type/Field format**. You can also enter 2 on several items and on the bottom of the screen, type: **LST\_CCPLT (field format)**.



To avoid tedious typing in relation to the number of fields to process using the previous procedure, you can create particular options to do that for you.

**Step 1** Use `AEDTLST LST_CSTST (FIELDS)` then press F2.

**Step 1** Enter 5 next to the Fields option list.

**Step 2** Select option 3=Copy, then option 2=Change.

**Step 3** Give it a different option name (beginning with a preferred letter). (ex.: R)

**Step 4** Press F4 on the `ACHGLSTE` command shown further below, and fill in the field format (ex.: ...R).

**Step 5** You can now use the new option R directly next to a field in the list in order to allocate the format.



**TRANSFORMING**

# 15 Working with the TRANSFORM menu: Propagation and Transformations

The following options of the **TRANSFORM** menu regroup all the ARCAD Transformer Field operations concerning propagation and transformation.

It is divided into several parts as indicated in the table(s) below:

| Option                    |                                       | ARCAD Command | Description  |
|---------------------------|---------------------------------------|---------------|--|
| Current environment       | Re-initialize the current environment | AINZCURENV    | These options allow you to enter an open version (and to verify the current version) before starting the propagation, modification and application functions in ARCAD Transformer Field. |
|                           | Display the current environment       | ADSPCURENV    |  |
| Configure the propagation | Work with propagation types           | AWRKPRPTYP    |  |
|                           | Work with standard routines           | AWRKSTDRTN    |  |

Table 15: Options to configure Transformer

Configure ARCAD Transformer Field using these two screens. For it to adapt to the propagation and/or automatic modifications, you require:

| Automatic Propagation and Transformation           | ARCAD command |
|--|---------------|
| Generate field names added to PF files             | ACVTDBFFLD    |
| Automatic modifications in the PF file's DDS       | ACVTDBFFLD    |
| Automatic propagation and modification of programs | ACVTPGMFLD    |
| Automatic modification of DSPF/PRTF DDS            | ACVTDDSFLD    |
| Work with transformer changes                      | AWRKFRMCHG    |
| Print propagation impact analysis                  | APRTFRMCHG    |
| Display derived fields of propagation              | ADSPDRVFLD    |
| Update the list of propagation limiters            | ASETPRPSTP    |

Table 16: Options for propagation and modifications

Here are two lists where you can find the 3 automatic transformation and propagation engines:

- List of fields impacted in the programs - LSTDRVFLD
- List of DB fields and DSPF/PRTF fields to be impacted - LSTDDSFLD

If you carry out a propagation without modifications, you will only start the `ACVTPGMFLD` (in steps 1 and 2). Then:

- `AWRKFRMCHG` will allow you to visualize the impacted components and the line details.
- `APRTFRMCHG` will allow you to print the impact analysis.
- `ADSPDRVFLD` will allow you to display the derived fields and especially, to go back through the propagation.
- `ASETPRPSTP` will allow you to set the propagation stoppers.

Lastly, you can visualize the two lists resulting from the propagation (`LSTDRVFLD` and `LSTDDSFLD`).

## 16 Detecting anomalies generated by propagation

The following are the lists of other DB fields where `ACVTPGMFLD` completed:

- Component/Field Warnings/Incident list
- Used propagation limiters list
- Components outside propagation list
- Unfound component's list (Pb read. source)

All these options are consultations via filters on the two lists resulting from the propagation (LSTDRVFLD and LSTDDSFLD).

|  | Options  | ARCAD Command           |
|--|--|-------------------------|
| Application of modifications/re-compilations | Apply the changes generated by ARCAD Transformer Field   | <code>AAPYFRMCHG</code> |
|  | Compile the modified + dependent objects                 | <code>CPLOBJDEP</code>  |
| Data recovery                                | Generate PF data field formatting programs               | <code>AGENFMTPGM</code> |
|  | Execute PF data field formatting programs                | <code>AEXCFMTPGM</code> |
| Other options                                | Remove flagging in a source                              | <code>ARMVMBRFLG</code> |
|  | Delete the info generated by a propagation               | <code>ACLRDRVFLD</code> |
|  | Convert a list of fields to a list of sources            | <code>ACVTLST</code>    |
| Special processes                            | List of DB fields with internal description in COBOL     | <code>AFLDDCTINT</code> |
|  | DB field list validation of internal description (COBOL) | <code>AFLDDCTIN2</code> |

Table 17: Other Transform options

# 17 Propagation types – AWRKPRPTYP

---

## Chapter Summary

|  |    |
|--|----|
| 17.1 Defining a propagation type.....                              | 70 |
| 17.2 Identifying propagation type and behavior.....                | 70 |
| 17.3 Defining automatic modifications at file level.....           | 74 |
| 17.4 Defining automatic modifications at screen/printer level..... | 80 |

The objective of propagation types is to define the behavior of the propagation and automatic modifications by ARCAD Transformer Field. A propagation type is characterized by:

- Its identifier which is a letter from A to Z; you will allocate its properties.
- The propagation types are regrouped in the **propagation groups**.

The propagation groups are coded in 3 characters. There is no relation between these and application codes.

The propagation group allows you to regroup one or several types of propagation. You can then use the same letter for the propagation type as the one used by someone else in another group. There can be no risk of confusion because you must specify the propagation group for each execution.

## 17.1 Defining a propagation type

---

To define a propagation type, you must fill in different information; however, it is not necessary to correctly fill in everything from the beginning. You can always come back later to complete some information (notably for the automatic modifications).

The different screens displayed are divided as follows:

- Identification of the propagation type and behavior of the propagation.
- Definition of automatic modifications at data file level.
- Definition of automatic modifications at screen and printer level.

## 17.2 Identifying propagation type and behavior

---

The group, the letter, and the title identify the propagation type. Only the 5 following parameters have an effect on the propagation's behavior.

When you start just one propagation without modification, concentrate only on these parameters.

All the parameters at the bottom of the 1st screen and the other screens concern the automatic modifications. Before starting the propagation with modifications, you must configure all the necessary parameters which follow.

```
AWRKPRPTYP          Work with propagation types          10/12/1 06:51:25
CSPWTRN1
Group. . . . . : AAA                                     1/5
Identification letter. . : A
```

Type or modify, then press ENTER.

```
Title. . . . . : For ACVTPGMFLD *DFT PRPMODE (*DBR/*ALLFLD/*RNMOBJ)
Propagation parameters
Type of propagation (Alpha. or Numer.) . . . . . : A      A, N, C
Number of characters to propagate. . . . . :      1      1...65535
Propagation in the multiplications/divisions . : N      Y, P, N
Propagation in the *LIKE DEFINE . . . . . : Y      Y, N
Propagation between programs via LDA . . . . . : N      Y, N
```

The following table shows the propagation parameters and other details:

| Propagation parameter | Description   |
|-----------------------|---|
| Propagation group     | <p>The propagation group is a 3-character code allowing you to regroup several types of propagations, which can be used simultaneously, into one group.</p> <p>It will be asked for when running the propagation engines.</p>   |
| Identification letter | <p>Required field. A letter from A to Z.</p> <p>The identification letter completes the propagation group in order to obtain the propagation type.</p> <p>This identification letter must be specified in the list of fields to propagate, in the field format as well as each of the fields to which the list will be attached giving the behavior to follow for the propagation and inherent modifications.</p> |
| Title                 | <p>The title corresponds to the title associated with the propagation type.</p>   |

*Table 18: Details of propagation parameters*

| Propagation parameter             | Description   |
|-----------------------------------|---|
| Type of propagation               | <div data-bbox="605 247 1299 409" style="border: 1px dashed gray; padding: 5px; margin-bottom: 10px;"> <p><b>Note</b><br/>A or N as a type of propagation is not totally synonymous with a numeric or alphanumeric field.</p> </div> <p>A : Alphanumeric. The propagation follows the pathway of all the characters to propagate. The derived fields will be impacted in the exact positions corresponding to the characters propagated. However, if a field in contact does not keep all the characters to propagate, it is not considered as a derived field. For more information, refer to <a href="#">Outlining propagation behavior on page 24</a>.</p> <p>N : Numeric. The propagation follows the pathway of the number of characters to propagate. The derived fields are entirely propagated if they are numeric. (They receive a field format equivalent to their length).The derived alphanumeric fields will be impacted to the exact positions corresponding to the propagated characters. For more information, refer to <a href="#">Outlining propagation behavior on page 24</a></p> <p>C : By character. The propagation is done independently for each character in the propagated field. In such a case, put 1 at the <b>Number of characters to propagate</b>. You indicate this value when doing transformation from alphanumeric to Unicode.</p> |
| Number of characters to propagate | <p>Enter the number of characters that follow the path to the core of the programs.</p> <p>For a numeric propagation, enter at least 1 or 2. Only those fields containing 1 or 2 digits of a field in contact will be considered as being derived fields. You can also enter a higher number (Ex. 5): in this case a field in contact that keeps 4 characters will not be propagated.</p>   |

*Table 18: Details of propagation parameters*



| Propagation parameter                    | Description  |
|--|--|
| Propagation in multiplications/divisions | <p>Specify if the propagation must be made in the calculating instructions: multiplication or division.</p> <p>Y: In multiplications and divisions, there is propagation of the factors to the results field (or vice-versa).</p> <p>P: There is propagation in the multiplications and divisions when it can be made without ambiguity, that is, towards the “product”. In multiplications, it will propagate from one of the factors to the product (results) but not vice-versa. In divisions, it will propagate from the results factor to the numerator only. Attention! The propagation is done from the divider to the numerator.</p> <p>N: In multiplications and divisions, there is no propagation of factors to the results field (or vice-versa).</p> <p>The choice between P and N is important according to the problem you are dealing with:</p> <ul style="list-style-type: none"> <li>• If you only want to carry out a propagation on the unitary prices, you should put ‘N’.<br/>Also with ‘UP MULT QTY AMOUNT’ -&gt; AMOUNT will not be propagated by UP.</li> <li>• If you want to carry out a propagation on everything given in amount, you should put ‘P’:<br/>Also with ‘UP MULT QTY AMOUNT’ -&gt; AMOUNT will be propagated by UP.</li> <li>• However, QTY will not be propagated (unless you put ‘Y’).</li> </ul> |
| Propagation in the *LIKE DEFINE          | <p>Specifies if the RPG, RPGLE, CBL, or CBLLE instruction *LIKE DEFINE is cause for propagation or not.</p> <p>Y: The definitions of a field in relation to another are cause for propagation (from factor 2 to the results factor or vice-versa). This means that this instruction is used to define the fields with the same length, and also the same nature concerning their contents.</p> <p>N: There is no direct propagation on the instructions *LIKE DEFINE between Factor 2 and the results Factor. This means that this instruction is used to define fields with the same length but whose contents are not necessarily of the same nature.</p>  |

Table 18: Details of propagation parameters

| Propagation parameter                | Description  |
|--------------------------------------|--|
| Propagation between programs via LDA | <p>The LDA is, sometimes, used as a method for parameter passage between called and calling programs.</p> <p>Y: There will be propagation via the LDA between the callers and the called (in both directions). The LDA division must be similar between the two calling programs.</p> <p>N: The propagated fields belonging to the LDA description do not set off propagation towards the callers (or the called).</p> |

Table 18: Details of propagation parameters

## 17.3 Defining automatic modifications at file level

The parameters situated at the bottom of screen 1/5, screen 2/5, and possibly 3/5 are linked to the automatic modifications you wish to carry out on the file fields. However, this will also cause modifications in the programs:

- In the case of field length extension, the working fields in the programs will also be extended.
- In the case of added file fields, you can ask for the insertion of routines for each read or write in the file.
- Also, if the file is described as internal in the programs, all the file descriptions will be modified.

You must first choose between the 4 types of modifications (Addition, Change type, Extension, or Deletion), then configure these modifications.

 **Reference**

For more information about the functions concerning this subject, refer to [Working with automatic modification functions on page 47](#).

Below, you will find the 3 parts of the screen concerned.

```

D.B. file field's modification parameters
Type of modif. (Add, Chg.type, Extension, Del.): E          A, C, E, D
New field length / Extension length . . . . . :      2    0..65535 (-nn..-1)
# decimals new / # decimals for extension . . :          0..9 (or -9..-1)
Type of new field (if different) . . . . . :          Blank or Type valid
Position of the modif. inside the field . . . :          0=End,-1=Str,1-
65535
More...
F3=Exit      F12=Cancel      F21=Command line
    
```

The above information concerns either the field extension values or the values of the newly-added file field. If you do not wish to carry out any file modification, leave the values at E=Extension and 0.

The following table shows the details of the DB file field's modification parameters.

| Parameter                                       | Description   |
|---|---|
| Type of modif. (Add, Chg.type, Extension, Del.) | <p>The modification to be carried out by the propagation on a database field (and the derived fields) can be of 4 different types:</p> <p>A: The <b>A</b>ddition of one (or two) fields in the files with correspondence to each field in the list of initial fields. To name the new fields, you need to first run the command to process the PF fields (<code>ACVTDBFFLD</code>) with the option <code>*NEWFLD</code>.</p> <p>C: The <b>C</b>hange type for fields.</p> <p>E: The <b>E</b>xtension of the length of each field from the list of initial fields.</p> <p>D: The <b>D</b>eletion of the field in the files, but also the deletion suggestions for the associated working fields as well as the display fields.</p> |
| New field length / Extension length             | <p>In the case of field addition, specify the total length the new field will have. Leave this value at 0 so that the new field has the same length as the previous one.</p> <p>When extending the length of a field, specify the extension length (increment) to add to the current length of the field. You can also place a negative value here if you wish to reduce a field.</p>   |
| # decimals new / # decimals for extension       | <p>In the case of a field addition specify the number of decimals for the new field. If the length of the new field is set at zero, its number of decimals will be identical to that of the previous field.</p> <p>When extending the length of a field, specify the number of decimals to add to the current number of decimals in the field.</p> <p>To only extend the decimal part of a field, specify the same value at the “Extension length and Nbr of decimals” parameters.</p>  |
| Position of the modif. inside the field         | <p>When extending this allows you to specify the exact position in the field where the extension must be. This can be useful if the modified field is divided into several sub-fields (not propagated independently).</p> <p>0 = End: The extension is made on the last character of the field. When adding a field to a structure, the field is added after.</p> <p>-1 = Start: The extension is made on the first character of the field. When adding a field to a structure, the field is added before.</p> <p>1-65535: You specify the exact position after which the new extension characters are inserted. A value from 1 to 65535 is incorrect here when adding a field.</p>   |
| Type of new field (if different)                | <p>When adding a field or changing the type of field, enter the type of the new field in the physical files only if it is different from the type of analyzed fields.</p> <p>If indicated, the type must be a valid field type in the DDS and PF. (A (Alpha) S (Extended) P (Packed) )</p>  |

Table 19: Modification parameters for DB file fields

On the following screen all the parameters concern a field addition, except the routine which can also be used in extension of the field's length.

```

AWRKPRPTYP          Work with propagation types          10/08/18
08:50:28
CSPWTRN1
Group. . . : AAA    Identification letter . . . . : A
2/5
  
```

```

Type or modify, then press ENTER.
In case of new field file . . . :
Position of addition in record . . . . . : *EOR          *AFTER, *BEFORE, *EOR
Dependent link with old field. . . . . : *NONE          *AFTER, *BEFORE,
*NONE
Similar functioning of the 2 fields. . . . : N            N=No, Y=Yes, A=Assign
Replace file field name in pgms. . . . . : N            Y, N
Routine inserted before writing file . . . : *NONE          Routine, *NONE (F4)
Routine inserted after writing file. . . . : *NONE          Routine, *NONE (F4)

If addition of the files from a 3rd field .:
Create a third field . . . . . : N                    Y, N
Third field length (if different). . . . . :              0..65535
Decimals # in third field. . . . . : 0                0..9
Type of third field (if different) . . . . :              Blank or Type valid
  
```

The following table presents the parameters of field addition.

| Parameter                          | Description  |
|------------------------------------|--|
| Position of addition in the record | <p>When adding a field, specify the exact position in the record where the new field must be added:</p> <ul style="list-style-type: none"> <li>*AFTER: The new field is added just after the analyzed field.</li> <li>*BEFORE: The new field is added just before the analyzed field.</li> <li>*EOR: The new field is added at the end of record.</li> </ul> <p>If several are processed for only one file, the added fields will be in the alphabetical order of the analyzed fields at the end of the recording.</p> |

Table 20: Field addition parameters

| Parameter                           | Description   |
|-------------------------------------|---|
| Dependent link with old field       | <p>When adding a field, specify if a hierarchical dependent link exists when the analyzed field is defined as a key for a file (physical or logical).</p> <p>*AFTER: The new field is added to the keys after the analyzed field.</p> <p>*BEFORE: The new field is added to the keys before the analyzed field.</p> <p>*NONE: No hierarchical link exists between the two fields. The new field is not added to the keys.</p> <p>This parameter has an impact on:</p> <ul style="list-style-type: none"> <li>• The automatic modifications of PF sources (by <code>ACVTDBFFLD</code>).</li> <li>• The automatic modifications of LF sources (by <code>ACVTPGMFLD</code>).</li> <li>• The automatic modifications of RPG or RPGLE sources (when the field is in key).</li> </ul>   |
| Similar functioning of the 2 fields | <p>When adding a field, indicate if the propagation must try to keep a similar type of functioning in the program processes.</p> <p>Three possible values:</p> <p>N: No similar functioning. The processing instructions are not copied.</p> <p>Y: Functions completely similarly. All the processing instructions are copied in the programs. The derived working fields will then receive the names of the associated fields in relation to each new field. Attention! The program will need to be modified.</p> <p>A: Similar functions for the assignment instructions only. Only the assignment instructions are copied in the programs. The derived working fields receive the names of the fields associated in relation to the new field.</p> <div data-bbox="451 1167 1295 1285" style="border: 1px dashed gray; padding: 5px; margin-top: 10px;"> <p><b>Note:</b> In RPG, the following are considered as assignment instructions: MOVE, MOVEL, MOVEA, Z_ADD, Z_SUB, PARM, DEFN.</p> </div> |

Table 20: Field addition parameters



| Parameter                              | Description   |
|--|---|
| Replace file field name in pgms        | <p>When adding a field, indicate if in the programs, you will use another field name in place of the analyzed field and added field. Processes will be added each time a file is accessed, in order to regroup the two fields after reading or to divide them before writing.</p> <p>This parameter is incompatible with similar functioning of two fields.</p> <p>Also, in this specific case, enter the appropriate parameters in screen 3/5.</p>   |
| Routine inserted before writing file   | <p>Specify the name of the standard code routine that will be added before each write to the file, to process the analyzed field.</p> <div style="border: 1px dashed gray; padding: 10px; margin: 10px 0;"> <p> <b>Reference</b><br/>For more information about standard routines, refer to <a href="#">Standard routines on page 86</a></p> </div> <p>Leave *NONE if no process needs to be added before writing to the file.</p> <p>The following names can be used in the source of the routine:</p> <p>\$\$F1: Indicates the name of the current analyzed field.</p> <p>\$\$F2: Indicates the name of the newly added field.</p> <p>\$\$F3: Indicates the name of the third field added to the files, or the name of the field group used.</p> |
| Routine inserted after reading file    | <p>Specify the name of the standard code routine that will be added after each read of the file to process the analyzed field.</p>  |
| Addition of the files from a 3rd field | <p>Specifying 'Y' in 'Create a third field' allows you to indicate that another field (3rd field) must be also be added to the files. To name this field, use the <code>ACVTDBFFLD</code> command with the option NEWFLD and the parameter THIRDFLD(YES).</p> <div style="border: 1px dashed gray; padding: 10px; margin: 10px 0;"> <p> <b>Note</b><br/>The utilization of this 3rd field is limited to file access; it can only be updated using routines inserted after reading or before writing.</p> <p>It is not concerned by similar functioning.</p> </div>   |
| Third field length (if different)      | <p>Specify the total length for the new field. Leave this value at 0 so that the new field has the same length as the analyzed field.</p>   |
| Decimals # in third field              | <p>Specify the number of decimals for this new field. If the length of the new field is set at zero, its number of decimals will be identical to the analyzed field.</p>  |

Table 20: Field addition parameters

| Parameter                          | Description   |
|------------------------------------|---|
| Type of third field (if different) | Enter the type of 3rd field in your physical files, only if it is different to the type of fields analyzed.<br>If entered, the type must be valid for a PF DDS type field. (A (Alpha) S (Zoned) P (Packed) ). |

Table 20: Field addition parameters

```
AWRKPRPTYP          Work with propagation types          10/08/18
09:11:32
CSPWTRN1
Group . . . : AAA      Identification letter . . . . : A          3/5
```

Type or modify, then press ENTER.

For the replacement of the file field names in the programs :


```
Prefix . . . . . : Valid character(s)
or Suffix. . . . . : Valid character(s)
# of recovery characters . . . . . : 0..3
Maximum field length limitation. . . . . : 0 or 5..10
```

For non-modifiable interface files :

```
Routine inserted before writing file . . . : *NONE      Routine, *NONE (F4)
Routine inserted after writing file. . . . : *NONE      Routine, *NONE (F4)
```

The first 4 parameters of this screen are only useful if you indicated **Y** at the replacement of file field names in the programs.

It allows you to enter the logic the `ACVTPGMFLD` (Step 3) is to follow at the automatic modification of programs, for giving a new name to the field file in the programs.

 **Example**

The analyzed field is called ARCOD. A field was added to the ARCOD2 files.

You want the 2 fields (after the file reading) to be placed end to end in a working field entitled XPCOD (the routine after reading will do this).

ARCOD will be renamed XPCOD throughout the program.

You just need to set up a rule to determine the name of these new program fields.

The following table presents the parameters for replacement of file field names.

| Parameter                              | Description                                     |
|--|---|
| Pgm replacement field Prefix or Suffix | Enter either the prefix or suffix but not both. |

Table 21: Replacing file field name parameters



| Parameter   | Description   |
|---|---|
| # of recovery characters                                | <p>Enter if the prefix (or suffix) must precede (or follow) the entire name of the analyzed field or overlap it.</p> <p>By using a value other than 0, you indicate the number of characters for the prefix or suffix that must overlap the start (or the end) of the analyzed field name.</p>  |
| Maximum field length limitation                         | <p>To build the name of the new field of the replacement fields in the pgms, enter the maximum length that the built field name can have.</p> <p>Specify a value from 5 to 10 for the maximum or enter 0: in this case the maximum possible length will be determined by the language of the programs that use this field. This will be 6 for RPG, RPG36, RPG38 and 10 for COBOL, CLP, RPGLE.</p> <p>If the built field name crosses this limit, there will be an overlay for the number of characters in excess.</p>   |
| Routine inserted for the non-modifiable interface files | <p>Specify here, the name of a standard code routine that will be added before each interface file write, to process the analyzed field.</p> <div data-bbox="526 793 1299 919" style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p> <b>Reference</b><br/>For more information about standard routines, refer to <a href="#">Standard routines on page 86</a>.</p> </div> <p>Leave *NONE if no process is to be added before writing the file.</p> <p>These routines are only used by <code>ACVTDDSFLD</code>.</p> <p>The following names are to be used in the routine source:</p> <ul style="list-style-type: none"> <li>• <code>\$\$F1</code>: Indicates the name of the renamed program field.</li> <li>• <code>\$\$F2</code>: Indicates the name of the file field.</li> </ul> <div data-bbox="526 1205 1299 1360" style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p> <b>Reference</b><br/>For more information about this subject, refer to <a href="#">Working with automatic modification functions on page 47</a>.</p> </div> |

Table 21: Replacing file field name parameters

## 17.4 Defining automatic modifications at screen/printer level

In the screen 4/5 and 5/5 you will set the behavior of automatic modifications for the DSPF/PRTF fields.

These parameters will be used by `ACVTDDSFLD`.

For more information on the functionalities concerning these modifications, first consult the automatic modifications of display and printer fields, and then refer to the following for learning how to set up the configuration.



```
AWRKPRPTYP          Work with propagation types          10/08/18
09:14:31
CSPWTRN1
Group. . . : AAA      Identification letter . . . . : A          4/5
```

Type or modify, then press ENTER.

```
For screens and printouts      :
Modification visible to the users . . . . : *ALWAYS      *ALWAYS, *IFFITS,
                                          *NEVER
Position of addition (if new field visible): *AFTER        *AFTER, *BEFORE
Routine inserted before screen display . . : *NONE        Routine, *NONE (F4)
Routine inserted after screen entry. . . . : *NONE        Routine, *NONE (F4)
Position of the hidden field in the format : *EOR         *EOR, *BOR, *NONE
For the name of the new field visible screen/status:
Prefix . . . . . : Valid character(s)
or Suffix. . . . . : Valid character(s)
# of recovery characters . . . . . : 0..3
Maximum field length limitation. . . . . : 0 or 5..10
```

The following table shows the parameters:

| Parameter                          | Description  |
|------------------------------------|--|
| Modification visible for the users | <p>Specify if the modification carried out on the database and in the programs should be visible to the users on their screens and in their printers.</p> <p><b>*ALWAYS:</b> In this case the propagated display or printer field visually undergoes the same modification as the one for the file fields. If you are using field extension, the display field is enlarged in the same way. If you are using field addition, a new field is added next to each display field. The added field is either a similar field whose name was determined by the propagation (\$\$A1 in the routines), or a new field whose name was built when the ACVTDDSFELD was run using the indications specified later on (\$\$F2 in the routines).</p> <p><b>*NEVER:</b> In this case propagated display or printer field does not visually undergo any modification. However, it is renamed and has a name made up for it when ACVTDDSFELD is run, from the indications specified later on (\$\$F2 in the routines).</p> <div style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p><b>Note</b><br/>However, despite this <b>*NEVER</b>, the displays can still undergo visual modification if you use <b>Add another field to the display/report</b>.</p> </div> <p><b>*IFFITS:</b> This is a mixed option where the choice between <b>*NEVER</b> or <b>*ALWAYS</b> is only made at the last moment according to the space available on the line. Note that this option is not available when adding a field as the routines to insert can be different between <b>*NEVER</b> and <b>*ALWAYS</b>.</p> |

Table 22: Screen/printer-level modifications parameters



| Parameter                                   | Description   |
|---|---|
| Position of addition (if new field visible) | <p>*BEFORE: If a new field is added visually to the display or printer it is placed before the propagated field.</p> <p>*AFTER: If a new field is added visually to the display or printer it is placed after the propagated field.</p>   |
| Routine inserted before screen display      | <p>Specify the name of a standard code routine that will be added before each screen display (or output of the printer format) to process the analyzed field, if it is in Output or Input-Output.</p> <div data-bbox="480 541 1299 667" style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p> <b>Reference</b><br/>For more information about standard routines, refer to <a href="#">Standard routines on page 86</a></p> </div> <p>Leave *NONE if no process is to be added before the screen display.</p> <p>The following names can be used in the routine source:</p> <p>\$\$F1: Indicates the name of the current analyzed field. This field will disappear from the screen when using the option *NEVER.</p> <p>\$\$A1: Indicates the name of the field associated to the analyzed field, when using the option “similar functioning of two fields” that are allowed to reach the display field (for field addition in the files).</p> <p>\$\$F2: Indicates the name of the newly added field in the screen, if generated by ACVTDDSFELD. This field is visible to the user.</p> <p>\$\$F3: Indicates the name of the third field added to the screens, if it is defined.</p> |
| Routine inserted after screen entry         | <p>Specify the name of the standard code routine that is added after each display reading to process the analyzed field, if it is in Input or Input-Output.</p>   |
| Position of the hidden field in the format  | <p>Specify the position where the hidden fields must be added to the screen format.</p> <p>*EOR: The hidden field is added at the end of the format.</p> <p>*BOR: The hidden field is added at the beginning of the format, after the other hidden fields. In this way it remains in place if, in SDA, you request the rearranging of the field format.</p> <p>*NONE: No hidden field is added to the format. The analyzed field is then defined in the usage program (on the added lines or on a code line already existing). This is the same for the associated field when it comes from propagation by similar processing.</p> <div data-bbox="480 1602 1299 1755" style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p> <b>Note</b><br/>For PRTFs, the option *NONE is used as default no matter what value was indicated here (because you cannot define a hidden field in a PRTF).</p> </div>   |

Table 22: Screen/printer-level modifications parameters

| Parameter                                   | Description  |
|---|--|
| Name of the new field visible screen/status | <p>ACVTDDSFELD will create a new display/printer field name using option *NEVER.</p> <div style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p><b>Note</b><br/>Even when using the option *ALWAYS for field addition, it is useful to anticipate this field as the name of this field cannot always be generated during the propagation.</p> </div> <p>You should simply give a rule to determine the name of these new fields.</p>  |
| Prefix or Suffix for the new visible field  | Enter either the prefix or the suffix, but not both.   |
| # of recovery characters                    | <p>Enter if the prefix (or suffix) must precede (or follow) the entire name of the analyzed field or overlap it.</p> <p>By using a value other than 0, you indicate the number of characters of the prefix or suffix that must overlap the start or the end of the analyzed field.</p>   |
| Maximum field length limitation             | <p>To build the name of the new field visible in the displays/printouts, indicate the maximum length the built field name can have.</p> <p>Specify a value from 5 to 10 for setting the maximum or indicate 0. In this case the maximum possible length will be determined by the program languages which use this display (or printout).</p> <p>In this way, it will be 6 for RPG, RPG36, RPG38 and 10 for COBOL, CLP, RPGLE.</p> <p>If the built name exceeds this limit, there will be a recovery for the number of characters in excess.</p> |

Table 22: Screen/printer-level modifications parameters

```

AWRKPRPTYP          Work with propagation types          10/08/18
09:17:59
CSPWTRN1
Group. . . : AAA      Identification letter . . . . : A
5/5
Type or modify, then press ENTER.

To add another field to the screen/status

Condition the addition of other field . . . . : N      N, I, O, B

Prefix . . . . . :          Valid character(s)
or Suffix. . . . . :          Valid character(s)
# of recovery characters . . . . . :          0..3
Maximum field limitation for field name. . . . :          0 or 5..10
Third field length (if different). . . . . :          0..65535
Decimals # in third field. . . . . : 0          0..9
Type of third field (if different) . . . . . :          Blank or Type valid

DDS Keywords for this field . . . . . :
    
```

You can ask `ACVTDDSFELD` to automatically add another field in the screens and printers (for each propagated field found).

Therefore, you must specify the following parameters as given in the table below:


| Parameter  | Description   |
|--|---|
| Condition the addition of other field                          | <p>Allows you to specify if you must add a new display for each display field (just before or just after), of a different format. (This field will be indicated by \$\$\$F3 in the routines.)</p> <p>For example, you can associate the currency code to each amount field.</p> <p>N: No "other fields" are added.</p> <p>I: One "other field" is added if the field is used in input (Input, Input-Output or Hidden Field).</p> <p>O: One "other field" is added if the field is used in output (Output, Input-Output or Hidden Field).</p> <p>B: One "other field" is systematically added.</p> |
| Name of the other field added in the screen/printer            | <p>You simply need to give a rule to determine the name of these new fields.</p> <div style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p> <b>Reference</b></p> <p>For more information about the new visible field's name, refer to the IBM i help. The principle here for naming this other field is the same.</p> </div>   |
| Characteristics of the added other field in the screen/printer | <p>Generally, it is necessary to specify the type and length of this other field, and to possibly assign DDS keywords to it:</p>  |
| Other field length   | <p>Specify the complete length this field will have. Leave the value 0 so that the new field has the same length as the analyzed field.</p>   |
| Nb. of decimals in other field                                 | <p>Specify the number of decimals this new field will have. If the length of the new field is set to zero, its number of decimals will be identical to the analyzed field.</p>  |

Table 23: Adding another field parameters

| Parameter                   | Description  |
|-----------------------------|--|
| Other field type            | <p>Enter the type of this field in the screens/printers, only if it is different than the type of the analyzed fields.</p> <div data-bbox="613 338 1297 495" style="border: 1px dashed gray; padding: 5px;"> <p><b>Note</b><br/>Be careful, if indicated, the type must be valid as a field type in the DDS of DSPF, PRTF. (A (Alpha) S (Signed) Y(Numeric), etc...).</p> </div>   |
| DDS keywords for this field | <p>You can define the keywords DDS of DSPF or PRTF which will be taken 'as is' in the DDS source for the newly added field.</p> <p>The syntax of these keywords must be correct.</p> <p>Only specify one keyword per line.</p> <p>The keyword will not be added if it is incompatible with the field utilization.<br/>Example: keyword VALUES(...) with a field in Output only.</p> <div data-bbox="613 793 1297 980" style="border: 1px dashed gray; padding: 5px;"> <p><b>Note</b><br/>You can specify a REFFLD(file field) keyword in order to indicate that the new field must be defined in relation to the reference file. This allows you to externalize certain checks.</p> </div> |

Table 23: Adding another field parameters

# 18 Standard routines

---

## Chapter Summary

|  |    |
|--|----|
| 18.1 Naming and storing standard routines .....        | 86 |
| 18.2 Using routines .....                              | 87 |
| 18.3 Ascertaining routine particularities .....        | 87 |
| 18.4 Using variables in standard routines .....        | 88 |
| 18.5 Checking language in standard routines .....      | 89 |
| 18.6 Inserting standard routines .....                 | 91 |
| 18.7 Working with standard routines – AWRKSTDRTN ..... | 92 |

The standard routines (also called FRM Routines) are the source parts which the user must write. Their objective is to carry out a particular process for a field (either a PF file field, or a display or printer field). The routine is described once in each language. It will be duplicated into several copies (by insertion to the existing sources), in order to repeat the process in many fields.

## 18.1 Naming and storing standard routines

---

Each routine has a name divided into 2 parts:

1. The first 5 characters constitute the name of the routine.  
For organizational purposes, try to give similar names in alphabetical order (RESA1, RESC2, RESC5) to all the routines that you create for special needs. In this way, they will be easier to find later.
2. The following characters (on 2 to 5 chars.) correspond to the language in which the routine was implemented. It is necessary for each routine to anticipate as much implementation as the language in which it can be used.

 **Important!**

In the case of a routine absent from a language, you can check to see if it exists in a near equivalent language (SQLRPG --> RPG, SQLRPGLE --> RPGLE, RPG38 --> RPG, RPT --> RPG, CLLE --> CLP ...). However, be careful in this case of slight syntax differences between these languages.

 **Note**

In an attempt to insert a source routine into a source, if you don't find the implementation for the language used, an error message is inserted into the source instead of the routine.

**ERROR:** Standard FRM Routine XXXXX not found for language YYYYYY.

All the routines are stored in the source file *AARFRTNF1* in the library produced by ARCAD.

## 18.2 Using routines

---

The following are the processes that use routines:

- a. `ACVTPGMFLD` Step 3: Automatic modifications of programs. In the case where the routines were configured to be inserted before file writing, or after file reading.
- b. `ACVTDDSFLLD`: Automatic modifications of screens and printers. In the case where routines were configured to be inserted before display, or after input.
- c. `AFMTFLDDTA` in `*GENPGM`: Generation of Recovery programs. You must describe in an RPGLE routine the process model that will allow you to generate recovery programs on the processed PF fields.

The main goal of routines is to simplify jobs. The routine must, therefore, be reliable. For that, first test if the code to be inserted meets the required functionality.

If in doubt, you can build a test (a few programs, screens, files), run the propagation with automatic modifications on your test, apply it, compile, make a few manual modifications and then test the obtained results.

If the testing is satisfactory, you can delete your test in the version using `option=99` ; the routine is now ready to be inserted to several copies in a number of sources.

## 18.3 Ascertaining routine particularities

---

Before devising a routine, it is important to understand its particularity. The points below present scenarios when an RPG, CLP or CBL routine will be in the code to be inserted.

### 1. RPG, RPGLE

If an RPG routine is in the code to be inserted (beginning with `BEGSR` up to `ENDSR`), it will only be inserted once in the program.

It is illogical to put the variables `$$F1`, `$$F2`, etc.... between a `BEGSR` and an `ENDSR` as it will only take the value of these variables.

Be careful when you visualize the new source (using `AWRKFRMCHG` and option 8); you may be surprised as the RPG sub-routine will be inserted right in the middle of processing but this gets fixed itself. During the application of modifications, it will be moved to the end of the code.

### 2. CLP

In the code to be inserted, you can also include CLP work variable declarations.

Each one of these variable declarations (beginning with `DCL`) will only be inserted once in a source (whether it comes from one or several routines); also, the declarations will be in place (after the PGM or the last `DCL`).

### 3. CBL

You can also place COBOL work variable declarations in the code to be inserted.

Each one of these variable declarations (beginning with a level number (01, 77, ...)) will only be inserted once in a source (whether it comes from one or several routines); also, the declarations will be in place (in the WORKING-STORAGE SECTION).

## 18.4 Using variables in standard routines

In this section you will see the list of FRM variables which can be used in standard routines. You can put these variable names either on a code line to be inserted, or on a line in routine check language.

In the source code describing the routines, you can use the following FRM variables; these will be replaced by the name of the true variables at insertion.

### Important!

In CLP, precede these variable names with &: &\$\$F1.

### Warning!

Don't forget to leave a few blanks after the variable name in the fixed column languages (RPG, RPGLE), because if the variable name is longer than the 4 characters of its \$\$xx code, you risk losing one or two characters.

**\$\$F1:** Name of the actual field in the file (PF, LF or DSPF/PRTF).

**\$\$A1:** For the generation of recovery programs, field name in associated file. For a DSPF/PRTF - name of the associated field if it was possible to name it during the propagation process. Otherwise it is blank.

**\$\$F2:** For a PF, LF name of the new file field. For a DSPF/PRTF - name of the field added to the screen (\*ALWAYS) or newly-renamed field in the screen replacing the previous field (\*NEVER).

**\$\$F3:** For a PF, LF name of the third file field, if it was defined. For a DSPF/PRTF - name of the "Other" field if you chose to add this other field in the configuration of propagation types (otherwise blank).

**\$\$FM:** Original field format. In this format, each character in the field is represented (max. 50 characters).

Be careful of the recovery programs; it is the format of the field as was entered in the `LST_CCPLT` field for the list of fields to propagate.

The format value is systematically inserted here between two apostrophes, so to test this value in a %IF you must also put the comparison value between two apostrophes.

The following table shows the format value of the field as entered in the `LST_CCPLT` field.

| Format Value | Description   |
|--------------|---|
| \$\$TY       | Type of original field for a character (A, P, S, ...).                            |
| \$\$LG       | Total length of the original field. (Ex: 5 or 15)                                 |
| \$\$LD       | Number of decimals in the original field (Ex: 0 or 2). 0 if it is an alpha field. |

Table 24: Field format values



| Format Value                       | Description  |
|------------------------------------|--|
| \$\$DF                             | Original file name (PF, LF, DSPF, PRTF) for the routine insertion.                                       |
| \$\$DT                             | Original file type for the insertion: PF(38), LF., DSPF., PRTF..   |
| \$\$DM                             | Original file format for the insertion.  |
| \$\$DP                             | Code insertion position in relation to the instruction. (A for After, and B for Before)                  |
| \$\$TX                             | Complete field text (only available for the routines used for generating recovery programs).             |
| \$\$T1,\$\$T2,\$\$T3,\$\$T4,\$\$T5 | Each one of the 5 parts of the text field, each on 10 characters (from 1 to 10, from 11 to 20, etc. ...) |

Table 24: Field format values

## 18.5 Checking language in standard routines

In the entered FRM routines, you can insert check instructions which allow you to direct the insertion processes. These instructions use a mini language based on several commands all starting with % and detailed below.

Once you start using a few lines from this check language, don't forget to run option 6 = Verify after the routine input. This will avoid incorrect insertions caused by test overlapping such as %IF.

| Language check command | Description   |
|------------------------|---|
| %CM condition          | Commentary - Each line having %CM is a commentary which explains the routine: it will not be inserted in the source; it is ignored by the check language. |

Table 25: Standard routine languages

| Language check command | Description   |
|------------------------|---|
| %EXIT condition        | Insertion end - Once an active line having %EXIT is reached, the insertion stops.   |
| %IF condition          | <p>Conditioning the Insertion - After the %IF you can place a test composed of two items to compare separated by a comparison operator; this could be: = &gt; &lt; &lt;&gt; != &lt;= &gt;= LIKE EQ GT LT NE LE GE.</p> <p>If the condition is verified, the lines placed after the %IF will be inserted into the source; otherwise the insertion will take place after the %ELSE or after the %ENDIF.</p> <p>If you want to test a value in relation to a blank, enclose the two items with apostrophes: %IF '\$\$A1' &lt;&gt; ' '.</p> <p>In comparison, the two values compared are put as uppercase letters before evaluating the test. (So, 'Aa' = 'AA' &lt;- Always TRUE.)</p> <p>For example, the LIKE operator is used with a comparison item in which the points are considered as any character. %IF \$\$FM LIKE 'CCCC...' indicates that the field format has 8 characters of which the first 4 are Cs and the last 4, any other character.</p> <p>The conditions can be interlinked (%IF in a %IF, etc..).</p> |
| %AND condition         | Following Conditioning - Allows you to have multiple conditions; the %AND must be placed on the line that follows an %IF, another %AND or an %OR. (%AND has priority over %OR).   |
| %OR condition          | Following Conditioning - Allows you to have multiple conditions; the %OR must be placed on the line that follows an %IF, an %AND or another %OR.  |
| %ELSE condition        | Negative Conditioning - Indicates the following lines are to be inserted only if the condition placed higher up is not verified.  |
| %ENDIF condition       | End of lines to insert for a condition - Indicates the end of a condition started higher up by an %IF. The %IF - %ENDIF must be matched.  |

Table 25: Standard routine languages

| Language check command | Description   |
|------------------------|---|
| %ONCE identifier       | <p>Insert once - Indicates that the following lines must be inserted only once in the source (the insertion place will be determined by the type of lines to insert (F, D, I, C Card) in RPG (LE).</p> <p>After %ONCE, you must specify a 3-character ID that will allow you to clearly identify these lines (in the case where they were already added (for another field), coming from the same or another routine).</p> <div data-bbox="469 558 1297 684" style="border: 1px dashed orange; padding: 5px;"> <p><b>Important!</b><br/>In these lines to be inserted once, do not use replacement variables; only their first value will be taken into account.</p> </div> |
| %ENDONCE               | <p>Indicates the end of a %ONCE block.</p> <div data-bbox="469 747 1297 974" style="border: 1px dotted lightblue; padding: 5px;"> <p><b>Note</b><br/>If you tend to write complex routines but you cannot anticipate all the cases (only the more frequent), you can arrange for all the non-anticipated cases to show up as compilation errors (therefore requiring manual intervention):</p> </div>   |

Table 25: Standard routine languages

## 18.6 Inserting standard routines

Standard routines are used in two very different cases. These are:

1. Generation of data recovery programs.
2. Insertion of processes after File read or before File write (or update). This action is configured in the propagation type.
  - You indicate by file, either the data files (PF, LF) or the Display/Printer Files (DSPF/PRTF).
  - The insertion is normally carried out just before or just after the order that starts an operation on the file, apart from the odd slight difference.

### 18.6.1 Insert routines in RPG – RPGLE

If the programs use the files with external descriptions, the routine will be inserted just before the write order (WRITE, UPDATE, EXFMT) or just after the read order (READ, READE, etc., CHAIN, EXFMT, CLEAR).

If the file is used as an external description of a data structure, processes are added before writing the \*DTAARA (IN) or after its reading (OUT).

If exceptional outputs are used (EXCEPT with except name), the routine will be inserted before the EXCEPT order if the field is in output (in O card).

If exceptional outputs are used (EXCEPT without except name), the routine will be inserted before the first EXCEPT order if the field is in output (in O card). You must verify the placement of the insertion and possibly move the inserted block.

If the file is managed in primary or secondary, the processes after read will be added to the start of C cards; the processes before update will be added before the Total C cards (L1, L2) or before the first subroutine.

It may be necessary to add conditions to these processes.

## 18.6.2 Insert routines in CLP

---

In CLP, only the `SNDRCVF` or `RCVF` can allow routine insertions.

## 18.6.3 Insert routines in COBOL

---

In COBOL, the inserted routines are also placed just before or just after the access to the file by READ, WRITE, REWRITE, START.

However, the orders for access to files are multi-line, therefore the insertion is made:

- after the line containing the end of instruction Dot '.' or the END-READ, END-WRITE.
- before the line containing the READ.

In COBOL, another method is often used to access files: the utilization of an intermediate BUFFER (like BUFFER is linked to the file, and BUF1 contains the field descriptions).

\* FILE READ, then MOVE BUFFER TO BUF1 a few lines further.

\* MOVE BUF1 TO BUFFER, then order WRITE BUFFER a few lines further.

In this case the insertion of routines is made just after or just before the MOVE order (if this is placed in the 50 lines of code after the READ or before the WRITE).

## 18.6.4 Insert no routines

---

No automatic modification is possible for the access to D.B. files via:

- SQL orders in SQLRPG or SQLCBL
- OPNQRYF or CPYF in CLP.

No routine will be inserted for these methods of file access. However, these orders are managed in the propagation and therefore, are marked.

## 18.7 Working with standard routines – AWRKSTDRTN

---

The `AWRKSTDRTN` command allows you to manage the standard routines. It displays the list of available routines. You can place yourself on screen from a specific name, and/or select the routines written for a

specific language.

In regards to the implementation of a routine for a language, you can enter one of the following options:

| Option   | Description   |
|--|---|
| 1 = Select   | Allows you to choose the routine (to indicate in the configuration, the propagation types).   |
| 2 = Edit   | Allows you to modify the routine contents with SEU. If you wish to modify the text, you can do this by exiting SEU then modifying the text and asking for it to be saved. |
| In COBOL, certain lines may be flagged as being in anomaly under SEU (as you only insert a bit of incomplete code). Also, the variables \$\$\$F1 are incorrect in syntax under this name. This will disappear when \$\$\$F1 is replaced by the real field name at the insertion into a COBOL source. |   |
| 3 = Copy   | Allows you to copy a routine to a new routine.  |
| 4 = Delete   | Deletes the routine in the language indicated.  |
| 5 = Display  | Displays the contents of the routine via SEU.   |
| 6 = Check  | This option allows you to verify if the FRM syntax for the Routine Check language present in the source is correct or not.  |

Table 26: Options to implement routines

A message will inform you if the command ran successfully i.e. without any errors. If there are errors, a spool file containing the routine with the annotated errors found is displayed.

**Important!**

The verification relies on the FRM syntax (%IF with %ENDIF, %ELSE with an %IF, %IF with a correct syntax condition, ...), but in no case will it verify the contents of the routines to insert to the corresponding language.

### 18.7.1 Set example 1 of a standard routine (Basic)

The objective of EZMFI routine to insert before file update and carry out a verification/conversion calculation of the original file field (\$\$F1) and build the 2 added file fields (\$\$F2 et \$\$F3); the calculation needs a number of decimals for the field (\$\$LD).

#### EZMFI routine in RPGLE

```

C*%CM=====
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C* Calculate ECART field and file field assignments before file updat
C          CALL          'CVZECRFI '
C          PARM          $$$F1          WCVTIN          30 9
C          $$F2          PARM          $$$F2          WCVTAL          30 9
C          $$F3          PARM          WCVTEC          30 9
C          PARM          $$LD          WCVTDC          2 0

```

**Note**

This RPGLE routine can also be a base for the generation of recovery programs.

**EZMFI Routine in CBL**

```
*%CM=====
* Lines automatically inserted by ARCAD-FRM - copyright ARCAD SOF
01 WCVT-EURO-GES      PIC S9(13)V9(5)  COMP-3.
01 WCVT-EURO-ALT      PIC S9(13)V9(5)  COMP-3.
01 WCVT-EURO-ECART    PIC S9(13)V9(5)  COMP-3.
01 WCVT-EURO-NBDEC    PIC 99          COMP-3.
*   Calculate ECART field and file zone assignments before file u
MOVE $$F1 TO WCVT-EURO-GES
MOVE $$F2 TO WCVT-EURO-ALT
MOVE $$LD TO WCVT-EURO-NBDEC
CALL "CVZECRFICB" USING WCVT-EURO-GES WCVT-EURO-ALT
WCVT-EURO-ECART WCVT-EURO-NBDEC
MOVE WCVT-EURO-ALT TO $$F2
MOVE WCVT-EURO-ECART TO $$F3
*%EXIT
```

## 18.7.2 Set example 2 of a standard routine (Complex)

The objective of EXAFA routine to insert before screen display to work with an alternating display (either in standard currency, or in alternate currency according to the value of the indicator 57).

The display field is replaced by a new \$\$\$F2 field in place of the previous one. A 3-character field is added next to it for informing the user of the currency display. The value of this currency code is not directly in the program; it must be loaded at the beginning of the process. Also, for certain fields, an \$\$\$A1 associated field containing the converted value is sometimes routed parallel to the data file until the screen.

The routine to insert is different if you have this associated field ('\$\$\$A1' <> ' ') or if it doesn't exist.

### EXAFA Routine in RPG

```

C*%CM=====
C*%CM Systematic addition of retrieval of values to test
C*%IF '$$$F3' <> ' '
C*%ONCE SRE
C*   Retrieval of values of currency codes for the Euro
C           EXSR SREURS
C*%ENDONCE
C           SREURS   BEGSR
C           CALL 'CVXCODES'
C           CALL 'CVXCODES'
C           PARM WDVGES 3           Stored curr.entered
C           PARM           WDVGEA 3           " " displ'd
C           PARM           WDVALS 3           Alt. curr. entered
C           PARM           WDVALA 3           " "
C           PARM           WDVALA 3           " " displ'd
C           ENDSR
C*%ENDIF
C*%CM=====
C*%CM Case of a known associated field (based on similarity).
C*%IF '$$$A1' <> ' '
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Sof
C* Assign the field either in stored currency, or in the other!
C           *IN57   IFEQ '0'           *IN57 OFF:sto.cur
C*%IF '$$$F3' <> ' '
C           MOVELWDVGEA   $$$F3           Display curr.code
C*%ENDIF
C           CALL 'CVZAFFIC'           Alternate display
C           $$$F1   PARM $$$F1   WCVTOU 309   Amount in sto.cur
C           $$$F2   PARM $$$A1   WCVTIN 309   Amount in alt.cur
C           PARM $$$LD   WCVTDC 20   Number of decimal
C           ENDIF
C*%ELSE
C*%CM Case of an unknown associated field.
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Sof
C* Assign the field either in stored currency, or in the other!
C           *IN57   IFEQ '0' *IN57 OFF:sto.cur
C*%IF '$$$F3' <> ' '
C           MOVELWDVGEA   $$$F3           Display sto. curr
C           Z-ADD$$$F1   $$$F2

```

```

C                                     ELSE
C*%IF '$$F3' <> '                    '
C                                     MOVE LWDVALA    $$F3                Display
C*%ENDIF
C                                     CALL 'CVZAFFIC'                Alternat
C          $$F1      PARM $$F1      WCVTOU 309 Amount in sto.cur
C          $$F2      PARM 0         WCVTIN 309 Amount in alt.cur
C                                     PARM $$LD      WCVTDC 20      Number of decimal
C                                     ENDIF
C*%EXIT
C*%ENDIF

```

**Note**

The call of the RPG sub-routine by EXSR SREURS is placed between %ONCE SRE and %ENDONCE; it will only be inserted once in the program. It may be necessary to move it (to put at the beginning of the program for example).

However, the contents of the SREURS routine (from BEGSR to ENDSR) will be, like all RPG sub-routines present in the standard routines code, inserted once in the program.

### EXAFA Routine in CBL

```

/*%CM=====
/*%CM  Routine:calculate Euro amount before display (with code in
/*%CM=====
*%IF $$DT = DSPF
*%OR $$DT = DSPF38
05 Screen indicator for function key Currency Change pressed
05  IN07                                PIC 1 INDIC 07.
88 IN07-ON                               VALUE B"1".
88 IN07-OFF                              VALUE B"0".
05 Screen indicator for display of message for Currency Change
05      57 OFF --> in stored currency  57 ON --> in alternate currency
05  IN57                                PIC 1 INDIC 57.
88 IN57-ON                               VALUE B"1".
88 IN57-OFF                              VALUE B"0".
*%ENDIF
01 Currency indicator for amounts
01 WCVT-CHOIX-AFF PIC X.
88 WCVT-CHOIX-AFF-GES                     VALUE B"0".
88 WCVT-CHOIX-AFF-ALT                     VALUE B"1".
01 WCVT-EURO-GES      PIC S9(13)V9(5)  COMP-3.
01 WCVT-EURO-ALT     PIC S9(13)V9(5)  COMP-3.
01 WCVT-EURO-NBDEC   PIC 99  COMP-3.
01 WCVT-DEV-GES-SAI  PIC XXX.
01 WCVT-DEV-GES-AFF  PIC XXX.
01 WCVT-DEV-ALT-SAI  PIC XXX.
01 WCVT-DEV-ALT-AFF  PIC XXX.
*%CM Systematic addition of retrieve processing of values to test
*%IF '$$F3' <> '                    '

```



```

*%ONCE SRE
*   Retrieval of values of currency codes for the Euro
CALL "CVXCODES" USING WCVT-DEV-GES-SAI WCVT-DEV-GES-AFF
WCVT-DEV-ALT-SAI WCVT-DEV-ALT-AFF
*   Basic : Activate the currency display in stored currency
SET WCVT-CHOIX-AFF-GES TO TRUE
*%IF $$DT = DSPF
*%OR $$DT = DSPF38
MOVE WCVT-CHOIX-AFF TO IN57 OF INDICATOR-AREA
* F7 Change displayed currency (applied after screen read)
IF IN07-ON
IF WCVT-CHOIX-AFF-ALT
SET WCVT-CHOIX-AFF-GES TO TRUE
ELSE
SET WCVT-CHOIX-AFF-ALT TO TRUE
END-IF
MOVE WCVT-CHOIX-AFF TO IN57 OF INDICATOR-AREA
GO TO FinEcrxxx
END-IF
*%ENDIF
*%ENDONCE
*%ENDIF
*%CM=====
*%CM Case of a known associated field (based on similarity).
*%IF '$$A1' <> '      '
* Lines automatically inserted by ARCAD-FRM - copyright ARCAD SOF
*   Assign the field either in stored currency, or in the other!
IF WCVT-CHOIX-AFF-GES
*%IF '$$F3' <> '      '
MOVE WCVT-DEV-GES-AFF TO $$F3
*%ENDIF
MOVE $$F1 TO $$F2
ELSE
*%IF '$$F3' <> '      '
MOVE WCVT-DEV-ALT-AFF TO $$F3
*%ENDIF
MOVE $$F1 TO WCVT-EURO-GES
MOVE $$A1 TO WCVT-EURO-ALT
MOVE $$LD TO WCVT-EURO-NBDEC
CALL "CVZAFFICB" USING WCVT-EURO-GES WCVT-EURO-ALT
WCVT-EURO-NBDEC
MOVE WCVT-EURO-GES TO $$F1
MOVE WCVT-EURO-ALT TO $$F2
END-IF
*%ELSE

```

**Note**

The comment lines which must stay as declaratives are preceded by a level N° (05). Without that, they will be inserted into the code (PROCEDURE DIVISION). However, you must manually replace this 05 by a comment\*.



05 Screen Indicator for function key Chng forced  
currency

05 IN07 PIC1 INDIC 07

## 19 Modifying D.B. files automatically – ACVTDBFFLD

### Reference

For more information about the possibilities offered, refer to [Working with automatic modification functions on page 47](#).

The ACVTDBFFLD command reaches two complimentary objectives, but which are very different:

1. The generation of names for fields added to the PF files (Option \*NEWFLD).
2. Automatic modification of PF files in DDS sources (Option \*CVTSRC).

This command is used file by file, format by format. It contains a parameter called **Recovery after abnormal end** that can be used (with option \*CVTSRC only) in case of an incident that interrupted the process. In this case only the field that was being processed at the moment of the incident is subject to a possible risk of double processing (partial or complete).

The instances are as follows:

- Propagation without modifications. No need to run ACVTDBFFLD.
- Length extension for PF fields or deletion of PF fields. You only use the ACVTDBFFLD command with the option \*CVTSRC. This will generate the PF source line modification propositions (visible by AWRKFERMCHG). ACVTDBFFLD can be run before or after ACVTSGMFLD or ACVTDDSFLD.
- Field addition in PF files

In this case, a precise chronology must be respected. Start ACVTDBFFLD option \*NEWFLD before starting the propagation, to name the new fields. ACVTSGMFLD needs to know the name of the new fields. However, the ACVTDBFFLD option \*CVTSRC can be started before or after ACVTSGMFLD or ACVTDDSFLD.

- Field addition: ACVTDBFFLD Option \*NEWFLD

The objective of this process is only to *give a name (and text) to the new field*.

It is necessary to run the process twice if you also add a third field to the files (Option THIRDFLD (\*YES)).

The method for finding new field names and text is as follows:

- Enter (in the FLDNAM parameter) from 1 to 10 possibilities for replacing the field names.
- Enter (in the FLDTEXT parameter) from 1 to 10 possibilities for replacing the field text.

At execution, ACVTDBFFLD attempts to apply the given criteria in order to offer the new field names.

To see the result: AEDTLST LSTxxxx (List of fields to process) LST\_CSTST(FIELDS).

The field name is situated in:

- LST\_CSST (Column entitled “Field Utilization”) for the added field.
- LST\_JZSEL2 (Column entitled “Selection field 2”) for the added 3rd field.

The new text is situated in:

- `LST_CTXT` (Text) in place of the previous one.

If you need to rerun the process several times (in a successive manner), indicate the name of the list with all the DB fields (LSTDBFFLD) in the option **FROMLST**. This will allow you to start from the original text each time.

- **Field addition: Manually name the fields**  
If you come across problems using the automation above (impossible to define reliable criteria) as you are not satisfied with the results, you can impose the new names yourself in the fields `LST_CSST` and `LST_JZSEL2`, and the new text in `LST_CTXT`.

You can use option 2 Change + F4, or also fill in these fields with the new field names with the help of one of your own programs.

## 19.1 Adding fields

---

### ACVTDBFFLD Option \*CVTSRC

This will generate propositions of new PF source lines for the new fields (visible by `AWRKFRMCHG`).

- The name of the new field will be taken in `LST_CSST`.
- The text of the new field will be taken in `LST_CTXT`.

If you also add a 3rd file field, then run the `ACVTDBFFLD` Option \*CVTSRC and Option **THIRDFLD(\*YES)**.

- The name of the third field will be taken in `LST_JZSEL2`.
- The text of the third field will be taken in `LST_CTXT`.

Be careful in this case. There could be problems for the field text (contained in `LST_CTXT` whether it is the 2nd or the 3rd field); the only solution in this case is to make copies of the list files (by `AEXTLST`):

- Make a copy for the list containing the names of field 2 and text of field 2.
- Make a copy for the list containing the names of fields 2 and 3 with the text from field 3.

With these 2 backup copies, you can easily restart the modification of PF sources, if necessary.

### ACVTDBFFLD Option \*RCLTXT

This option is only used for refreshing the text of each field in the list by retrieving the original text from another list, to be indicated in the \*FROMLST parameter (normally the LSTDBFFLD).

## 20 Propagating and modifying programs automatically – ACVTPGMFLD

### Reference

For information about the possibilities available in propagating and modifying programs automatically, refer to:

- [Working with automatic modification functions on page 47](#)
- [Working with propagation functions on page 22](#)

The `ACVTPGMFLD` command starts the conversion and propagation analysis on the program sources from a list of fields to process.

It stores the flags and modifications in the *FRM modifications* file, accessed using `AWRKFRMCHG` and then validated by the `AAPYFRMCHG` command.

This command works by processing the fields in their order of appearance and in no case program by program. This is why it does not contain the **Recovery after abnormal end** parameter. The process of each step (1, 2, or 3) is indivisible. However, note that step 3 works source-by-source.

The following table shows the different parameters of the `ACVTPGMFLD` command.

| Parameter                                  | Description   |
|--|---|
| Propagation group                          | Enter the code of an existing propagation group.<br>Use the command <code>AWRKPRPTYP</code> to work with propagation types and to see the groups already defined.   |
| List of impacted fields (LIST)             | Indicates the name and library of the file of the list containing the fields to process for which you want to carry out the propagation and modifications.  |
| List of propagation monitoring (LSTDRVFLD) | Indicates the name and library of the list file that will receive the derived fields obtained by the propagation. This file will contain one copy by field / program couple.<br>The name of this list (with its library) identifies the propagation. You can delete the results of a propagation from the ARCAD database using the command <code>ACLDRVFLD</code> on this list. |
| List of D.D.S. fields (LSTDDSFLD)          | Indicates the name and library of the list file that will receive the different externally described fields obtained by the propagation.<br>In this file you will find both the fields of the physical files (PF) and logical files (LF) as well as the display fields (DSPF) and the printer file fields (PRTF).   |
| To development env. (TOENV)                | Indicates ONE or SEVERAL modifications and flagging destination environments. It is however, possible to carry out propagation for the fields of a file by notifying the applications concerned.<br>For every destination environment you must specify the n° of an open version.   |

Table 27: ACVTPGMFLD command parameters

| Parameter                         | Description  |
|-----------------------------------|--|
| Steps to execute (CVTSTEP)        | <p>Indicates if the propagation is to be done in its entirety or if you just want to run 1 or 2 steps.</p> <ul style="list-style-type: none"> <li>*ALL - The 3 propagation steps are run successively:               <ol style="list-style-type: none"> <li>Entry of the components.</li> <li>Propagation from field to field.</li> <li>Program (and LF) modifications.</li> </ol> </li> <li>1 or 2 or 3 - You only run one step.</li> <li>1-2 or 2-3 - You run two successive steps.</li> <li>1 or 2 or 3 - You only run one step.</li> <li>1-2 or 2-3 - You run two successive steps.</li> </ul> <p>This possibility to run the process in steps follows certain rules:</p> <ul style="list-style-type: none"> <li>You can only begin one step if the previous step was executed.</li> <li>You can rerun the propagation step 2 even when it has already been executed. Remember to delete the flagged lines beforehand though (<a href="#">AWRKFRMCHG</a>).</li> </ul> <p>It is imperative that <b>all the components</b> which the propagation relies on, haven't evolved between 2 steps.</p> |
| Literal conversions (CVTLIT)      | <p>Indicates if the modifications must be carried on the literals in relation to the fields concerned by the propagation.</p> <p>*YES: The literals are modified.</p> <p>*NO: The literals are not modified.</p> <p>This option is presently inactive. However, you can consult the set of propagated literals (even if you answered *NO for this parameter).</p>  |
| Warning if field already enlarged | <p>Only useful in the case of field extension (and in an alphanumeric type propagation).</p> <p>Specify if you want to consider the detection of fields which have already been extended, as an anomaly Warning. The fields concerned have the number extension characters at the beginning followed by the characters to propagate.</p> <p>*YES: These fields are considered in anomaly: the MSG1436 error message will be present in the list obtained by the <a href="#">APRTFRMCHG</a> command. These fields will not undergo length modification.</p> <p>*NO: This anomaly will not be detected.</p>  |
| Program to process (PGM)          | <p>Indicates if the propagation must be carried out on all components of the application (*ALL) or is limited to one specified program.</p> <p>You can also indicate *LST to specify the exact list of components that can be propagated. In this case, specify the name of the list at the LSTPRPPGM parameter.</p>   |

Table 27: ACVTPGMFLD command parameters

| Parameter                                 | Description  |
|---|--|
| List of pgms to propagate (LSTPRPPGM)     | <p>Indicates the name and library of the list file, list of programs to propagate (if the PGM parameter is at *LST).</p> <p>This list must have a source member list format.</p>   |
| List of propagation stoppers (LSTPRPSTP)  | <p>Indicates the name and library of the list file containing the fields and/or programs not to be taken into account in the propagation (propagation stoppers). This list is created empty when the <code>ACVTPGMFLD</code> command is first run. If you encounter any problems during the propagation, you can specify the names of programs or fields not to be processed using the command <code>ASETPRPSTP</code>.</p>  |
| Examine internal descriptions (CVTINTDSC) | <p>This parameter indicates that the propagation-modification process must also check the internal descriptions of database files (PF &amp; LF) and the display (DSPF) and printer (PRTF) files.</p> <p>In this case, it will examine in RPG(LE), the I and O cards corresponding to all the files. In COBOL, it looks at the internal format descriptions. In CLP, it mainly deals with CPYF orders and the sorting based on a positional description of the record fields.</p> <div data-bbox="500 814 1295 932" style="border: 1px dashed gray; padding: 5px; margin: 10px 0;"> <p><b>Note</b><br/>For applications in pure 36 mode (RPG36 &amp; RPT36) it is completely useless to answer *YES at this parameter.</p> </div> <p>*NO: It is a pure native IBM i application. The database files and display and printer fields are not described internally in the programs.</p> <p>*YES: It is a “mixed” application. The database files, the display fields or printer fields can sometimes be described internally in the programs.</p> <p>In this case the propagation and modification process will be slowed down because it will examine the internal descriptions in the programs for each processed file (by making a link with a possible substitution of the file name (OVRDBF) carried out in a CLP calling the program (internal file name different to a database file name)).</p> <p>The objects of the physical and logical files online must be those before modification (normally the ones from the repository) as it is from the description of these that <code>ACVTPGMFLD</code> finds the length and position of fields. The files must not have already been compiled in the version with the new descriptions.</p> |
| Propagation from Pgm to Pgm (PGMPGMPPR)   | <p>Indicates if propagation must be carried out from one program to another when one concerned field is used as a call or receive parameter.</p> <p>*YES: The propagation follows the fields that are program parameters.</p> <p>*NO: The propagation is not carried out from one program to another.</p> <p>Irrespective of the value of this parameter, the source copies (COPY/Clause...) are processed.</p>  |

Table 27: ACVTPGMFLD command parameters

| Parameter                           | Description  |
|-------------------------------------|--|
| Propagation trace by field (PRPTRC) | <p>Indicates the storage level for propagation traces.</p> <p>*FIRST: For each propagated position, you store only the first cause; this allows you to return through the propagation history log from any propagated field up until the original field list by using the command <code>ADSPDRVFLD</code>.</p> <p>*ALL: Stores all the links between fields. This possibility can be used with ARCAD Observer to graphically view the different paths followed for the propagation. However, these different links cannot be questioned using the native command <code>ADSPDRVFLD</code>: only the link seen first (first cause) will be available.</p> <div data-bbox="500 579 1299 699" style="border: 1px dashed gray; padding: 5px; margin-top: 10px;"> <p><b>Note</b><br/>The graphic interface in ARCAD Observer allowing this interrogation is not available at the time of writing.</p> </div> |
| Print anomaly lists (PRTERR)        | <p>Indicates if the propagation (Step 2) must print the detected anomaly lists.</p> <p>*YES: The anomaly lists are printed (placed in spool).</p> <p>*NO: The anomaly lists are not placed in spool. You can, however, consult (and print) them via the TRANSFORM menu.</p>  |

Table 27: ACVTPGMFLD command parameters



## 21 Modifying DDS automatically – ACVTDDSFLD

### Reference

For more information about the available possibilities, refer to:

- [Working with automatic modification functions on page 47](#)

This command works file by file, format by format. It contains the **Recovery after abnormal end** parameter which can be used in case of any incident having interrupted the process. In this case only the display or printer (or the file) which was being processed at the moment of the incident, is subject to a possible risk of double processing (partial or complete, routine added twice).

The `ACVTDDSFLD` command starts the modification engine of D.D.S. fields contained in the DSPF & PRTF, as well as the addition of conversion routines in the programs when reading or writing D.D.S. fields (DSPF, PRTF or database file fields that were non-modified but concerned by the propagation (Interface files)).

It stores the proposed changes in the *FRM modifications* file which is visible with the command `AWRKFRMCHG` and then be validated by `AAPYFRMCHG` command.

The `ACVTDDSFLD` command exploits the list file of D.D.S. fields obtained by the command `ACVTPGMFLD` which must have been run beforehand.

The following table presents the different parameters of the `ACVTDDSFLD` command.

| Parameter                       | Description  |
|---------------------------------|--|
| Propagation group (PRPGRP)      | Enter the code of an existing propagation group.<br>Use the <code>AWRKPRPTYP</code> command to work with the propagation types in order to know the defined groups.  |
| List of D.D.S. fields (LIST)    | Indicates the name and library of the list file containing the different fields with external descriptions obtained by the propagation (D.D.S. field file).<br><br>Note that it is possible to carry out different types of display/printer modifications according to the display fields and prints. For that, subdivide the list into as many lists as necessary and then run several processes separately by modifying the configuration of the propagation types.<br><br>However, it is better that all the fields of a display format (or PRTF) are present in the same list and therefore processed in one single operation. |
| DSPF/PRTF processing (DSPFPRTF) | This is the most common case for the <code>ACVTDDSFLD</code> engine. It specifies if you want to carry out the process of automatic modifications for the DSPF/PRTF (with the programs that use them).<br><br>*YES: The DSPF and PRTF will be modified according to the display/print configuration defined in the propagation types. For the "PRTF36" that doesn't exist as a component, only the programs will be modified.<br><br>*NO: No process will take place for the display/printer fields.   |

Table 28: ACVTDDSFLD command parameters

| Parameter                             | Description   |
|---------------------------------------|---|
| Interface file processing (INTERFACE) | <p>This is a particular case where ACVTDDSFLD can be used to process the interface file fields detected by the propagation.</p> <p>This sets off the addition of conversion routines in the programs for the database fields not modified but concerned by the propagation (interface files, for example).</p> <p>It is important to note that only the non-modified DB fields that don't belong to a modified file or screen will be processed that is, those having the MSG1435 warning message in the APRTRFMCHG list.</p> <p>This corresponds to the fields found in the anomalies list (DB Fields in which ACVTSGMFLD ended).</p> <p>*YES: When necessary, the routines defined in the propagation group (specifically for the interface files) will be added in the programs after the reading of and before the writing of, database files.</p> <p>*NO: No conversion routines will be added for these files. They will be ignored during process.</p> |
| Shorten the text (SHORTENTXT)         | <p>Specifies if you want to allow ACVTDDSFLD to shorten the text of a constant appearing in the display or print line in the case where there isn't enough room to carry out an extension or addition of a field. It is the closest constant or by default the first constant long enough that will be shortened (its last characters will be erased).</p> <p>*YES: If necessary, the constants (hard coded in the DDS or in the form of text coming from the message ID) can be shortened.</p> <p>*NO: Text shortening for constants is forbidden.</p> <p>In *IFFITS, the process automatically switches to *NEVER once there are no more blanks available on the line. In *ALWAYS the fields will be added or extended setting off a character overlap, with a warning message.</p>   |

Table 28: ACVTDDSFLD command parameters

| Parameter                               | Description   |
|---|---|
| To development environment (TOENV)      | <p>Indicates the destination environment for modifications and flags.</p> <p>If you proceed with a multi-application propagation, it is necessary to divide the list of DDS fields into as many lists as needed (one per application) and to start the ACVTDDSFLD process for each list obtained in each application.</p>   |
| List of impacted fields (PFFLDLIST)     | <p>Specify the file name and library for the list containing the different PF fields at the origin of the propagation.</p> <p>This information is only used in the case where a display (or printer) field is defined in REFFLD in relation to a physical file (or reference file) field.</p> <p>If the processed field is visibly extended and the field in REFFLD is present in the list of PF fields to process, it will be implicitly extended (without modifying the DDS).</p> <p>If the processed field is visibly extended but the field in REFFLD is absent from the list of PF fields to process, it will be relatively extended (+n in the length and possibly in the number of decimals).</p> <p>If a new field is added in the display, it will be defined in relation to the new field in the physical file (when the correspondence is possible).</p> <p>Note - If you leave this parameter at *NONE, it considers that the fields in REFFLD are processed by the same type of modifications.</p> |
| Propagation follow up list (DRVFLDLIST) | <p>Specify the file name and library for the propagation follow up list generated by ACVTPGMFLD.</p>  |

Table 28: ACVTDDSFLD command parameters

## 22 Modifying FRM – AWRKFRMCHG

The `AWRKFRMCHG` command allows you to work with the flags and automatic modification propositions generated by the following commands:

- Automatic modification in PF sources – `ACVTDBFFLD`
- Propagation and automatic modification in the programs – `ACVTPGMFLD`
- Automatic modification in the DDS of the PRTF/DSPF – `ACVTDDSFLLD`

These flags and modifications are stored in the propagated source line file. You gain access to the management of all the impacted components by the automatic transformation and analysis tools of these modifications. One line is displayed per impacted source.

Certain components are in warning (with `WTG W` in the status field) which means their processing caused at least one anomaly detected by the propagation.

**Note**

Once a line is displayed here for a component, it is no longer possible to check it out in another version (for modification). In this case you must delete the lines from `AWRKFRMCHG` for this component (to release it).

The following table presents the different parameters of the `AWRKFRMCHG` command.

| Parameter                     | Description  |
|-------------------------------|--|
| Environment (ENV)             | You can specify here, the environment on which you wish to run the command. It corresponds to the destination environment indicated at the execution of the engines.   |
| Modify (PGMR)                 | Allows you to pre-select the components to display in relation to the specified modifier.<br>*ALL: All the impacted components of the application will be displayed, whoever the author of the change request may be.<br>*: Only the components relative to the modifications requested by the profile will be displayed.<br>Name: You can specify the profile name for which you want the list of components concerned. |
| Type of modification (CHGTYP) | Allows you to select the items in the list according to the type of status that characterizes them.<br>*WTG: The selected items will be those waiting on the modification application process by the <code>AAPYFRMCHG</code> command.<br>*ALL: All the items of the list will be displayed.  |

Table 29: AWRKFRMCHG command parameters

The following table displays the options available for each impacted source.

| Option                     | Feature  |
|----------------------------|--|
| Option 4=Undo Change       | <p>The deletion of all the modification propositions for the component. By F17=Delete all, the option 4 is placed next to all the components and by confirming, you will delete all the flags and modification propositions (necessary if you want to restart the propagation).</p> <p>This option is only authorized for the source modifications allocated to your user profile.</p>   |
| Option 5=Source            | <p>Allows you to visualize the source of the component before modification.</p>  |
| Option 6=Print FRM Modifs. | <p>Starts the printing of flagged or modified lines for the component. (See: <a href="#">APRTFRMCHG</a>.)</p>  |
| Option 8=New source        | <p>Allows you to visualize the new source as it will be after the modifications are applied. (See: <a href="#">APRTFRMCHG</a>)</p>   |
| Option 10=Line details     | <p>Displays all the flagged code lines, added, or modified for the component.</p> <p>In the screen displayed, two options are available at this level:</p> <p>4=Delete: On a line modification (can only be used if *WTG is in the CHGTYP parameter of the command <a href="#">AWRKFRMCHG</a>).</p> <p>6=Message Warning: On the lines in warning (with a W before the source type, you can obtain the corresponding warning message).</p> |
| Option 11=Derived fields   | <p>With this option, you will reach the screen displaying the derived fields (<a href="#">ADSPDRVFLD</a>).</p>   |
| Option 23=References       | <p>Useful for interrogating the component cross-references.</p>  |

Table 30: Options for impacted sources

## 23 Printing the impact analysis – APRTFRMCHG

The `APRTFRMCHG` command allows you to edit or display the impact analysis results obtained by the propagation and the automatic modifications started with the `ACVTxxxFLD` commands.

You can request a detailed edition if desired, by member or by member and field, with or without the source lines concerned by the propagation (in their previous form and / or in the new form after automatic modification), possibly with all the other source lines.

The edition includes:

- a term indicating the type of flagging or modification.  
(MRK. ., MRKDEF, CHGDEF, NEWDEF-I, etc....)
- the propagated fields (and the literals) are underlined (or over-scored).
- the lengths (or positions) of these fields are also underlined (or flagged at the display by some +++++).

If anomalies were detected for the component, messages appear under each line concerned.


The following table presents the different parameters of the `APRTFRMCHG` command.

| Parameter                             | Description   |
|---------------------------------------|---|
| List of propagation monitoring (LIST) | Indicates the name and library of the propagation list file obtained from the <code>ACVTxPGMFLD</code> command.<br><br>If you enter an incorrect or missing name of a file list, the display can be carried out nevertheless if you gave the name of a member to be processed. However, you will lose the visualization of underlining the field names in the source code lines.  |
| Editing Level (LEVEL)                 | Specifies if you must edit the impact analysis and the modifications by source member or by member source / field couple.<br><br>*MBR: The editing is classed by source member then line n°.<br><br>*FLD: The editing is classed by member / field. For every field impacted in a member, you will find the flagged and modified lines.   |
| Source line printout (SRCLINES)       | Specify if you must edit the details of flagged or modified source lines.<br><br>*NONE: The source lines are not edited.<br><br>*OLD: The flagged or modified source lines are edited in their previous form before modification.<br><br>*NEW: The flagged or modified source lines are edited in their new form, after modification.<br><br>*BOTH: The flagged source lines are edited once only. However, the modified source lines are edited in two forms: before and after modification. |

Table 31: APRTFRMCHG command parameters

| Parameter                       | Description  |
|---------------------------------|--|
| With all the lines (OTHLIN)     | In the case where you have requested the editing of modified source lines and where the editing is done by member/source line n° (*MBR), this allows you to request the editing of other source lines not concerned by the propagation.<br>*NO: Only the lines concerned by the propagation are edited.<br>*YES: All the source lines are edited.  |
| Member (OBJ)                    | This parameter allows you to limit the editing of the propagation impact analysis for a specific source member. In the opposite case, leave the value at *ALL.   |
| Source member type (TYPE)       | Enter the type of the source member here (after having specified a source member name at the parameter OBJ). You can also leave the value *ONLY in the case where there is no ambiguity.   |
| Output (OUTPUT)                 | Specify the type of output you wish for consulting the information given by this command.<br>* : The output is carried out on screen. In this screen, the impacted fields will be “over-scored.”<br>*PRINT: The generation of a spool for the requested information (the impacted fields will be underlined).<br>You can consult the list by using the display function for job spool files (WRKSPLF command). |
| To development environ. (TOENV) | Indicates the destination environment for modifications and flags.   |

Table 31: APRTFRMCHG command parameters

 **Example**  
The example of flag and modification propositions on a source is shown below.

APRTFRMCHG

```

                                     Impact analysis of propagation list
Propagation list. . . . . : FRUD1.53.A / LSTDRVFLD
Print source lines. . . . . : *BOTH
Application/Ref./Version . . . : FRU / 0 / V 1.00.A
    
```

| Member   | Type | Total No<br>Field | Specific No.<br>specifics | Specific No.<br>flagged | Specific No.<br>modified | Specific No.<br>added |
|----------|------|-------------------|---------------------------|-------------------------|--------------------------|-----------------------|
| ASS010R1 | RPG  |                   | 286                       | 30                      | 13                       | 348                   |

| Field<br>decimals | Total No<br>of Specifics | Specific No.<br>flagged | Specific No.<br>modified | Specific No.<br>added | Specific No.<br>Length |
|-------------------|--------------------------|-------------------------|--------------------------|-----------------------|------------------------|
|                   | 286                      | 30                      | 13                       | 348                   | Interro client         |

```

for dep
100 H D J
200 *-----
300 * Client interrogation to know if there

is a command
400 * excess
500 * with the possibility to modify the

insurance amount
600 *-----
700 *
800 FCLIENTP1UF E K DISK
900 FARTTARP1IF E K DISK
1000 FASS010D1CF E WORKSTN
1100 F*
1200 E MES 1 4 70
1300 E QTY 3 8 0
MRKDEF 1400 E PU 4 9 3
NEWDEF-I 1400 E P9 4 9 3
1500 E*
1600 I* quantity table.
1700 I DS
1800 I 1 240QTE
1900 I 1 80ZRTQT1
2000 I 9 160ZRTQT2
2100 I 17 240ZRTQT3
2200 I* unitary price table.
2300 I DS
MRKDEF 2400 I 1 363PU
NEWDEF-I 2400 I 37 723P9
MRKDEF 2500 I 1 93ZRTPU1
NEWDEF-I 2500 I 37 453ZRTPA1
MRKDEF 2600 I 10 183ZRTPU2
NEWDEF-I 2600 I 46 543ZRTPA2
MRKDEF 2700 I 19 273ZRTPU3
NEWDEF-I 2700 I 55 633ZRTPA3
MRKDEF 2800 I 28 363ZRTPU9
NEWDEF-I 2800 I 64 723ZRTPA9
2900 I*
3000 I* DS for inversion date.
De CHGDEF 3100 IDSJMA DS 6
à --> CHGDEF 3100 IDSJMA DS 8
3200 I 1 20DSJMAJ
3300 I 3 40DSJMAM
De CHGDEF 3400 I 5 60DSJMAA
à --> CHGDEF 3400 I 5 80DSJMAA
De CHGDEF 3500 IDSAMJ DS 6
à --> CHGDEF 3500 IDSAMJ DS 8
De CHGDEF 3600 I 1 20DSAMJA
à --> CHGDEF 3600 I 1 40DSAMJA
De CHGPOS 3700 I 3 40DSAMJM
à --> CHGPOS 3700 I 5 60DSAMJM
De CHGPOS 3800 I 5 60DSAMJJ
à --> CHGPOS 3800 I 7 80DSAMJJ
3900 I*

```



```

4000 C*-----*
4100 C*   Keys
4200 C*-----*
Euro  INS*ONCSRE 4300 C*   Retr. of currency code values for
      INS*ONCSRE 4300 C           EXSR SREURS
      4300 C           KTAR       KLIST
      4400 C           KFLD       ZCOART
De    CHGDEF    4500 C           KFLD       WDATAR
60
à --> CHGDEF    4500 C           KFLD       WDATAR  80

. . . . .

6200 C*-----*
6300 C*   Initialization complete
6400 C*-----*
6500 C           ACTECR       IFEQ 'INI1'
6600 C           CLEARFMT01
FRM  -  INS*AFTER 6600 C*   Automatically inserted lines by ARCAD-
field, or  INS*AFTER 6600 C*   Allocate either management currency
      INS*AFTER 6600 C           CALL 'CVXSAICO'
      INS*AFTER 6600 C           PARM $MNTAS       WCVTZE
309
      INS*AFTER 6600 C           PARM CMNTAS       WCVTDV
3
      INS*AFTER 6600 C           ZMNTAS       PARM ZMNTAS       WCVTOU
309
      INS*AFTER 6600 C           ZMNTA9       PARM ZMNTA9       WCVTIN
309
      INS*AFTER 6600 C           PARM 0           WCVTDC
20
      6700 C           MOVEL 'AFFI'       ACTECR
      6800 C           ENDIF
      6900 C*-----*

```

Copyright ARCAD Software

## 24 Displaying derived fields – ADSPDRVFLD

---

The `ADSPDRVFLD` command allows you to display the fields derived from the propagation, but the most interesting part of this is to be able to know the path the propagation took to reach a certain field.

When using `ADSPDRVFLD`, you must give the name of the derived fields list (`*CURENV/LSTDRVFLD` is used by default). Then, you can:

- a. Display the list of a program's derived fields.
  - By putting `*` for the field name and by entering the name of the program.
- b. Display the list of programs containing a derived field.
  - By putting `*` for the program name and entering the field name.
- c. Lastly, and most importantly, go back up the path that the propagation followed in order to reach a program field, this time entering the field name and the program name.

In displays a) and b), the program field or field programs display is carried out alphabetically (with no relation to the chronological order of the propagation).

For each field, you find the following information using the **F11** function key:

- Field name.
- Name and type of program.
- Source line extract where the field is found before causing the first propagation to the line field.



### Reference

For more information about source lines, refer to [The propagation layout on page 31](#).

- Field format (Format code of compressed field).
- Recording N° in the `LSTDRVFLD` (corresponding to the propagation chronology).

## How and when to use these screens

You may need to go back up the path followed by the propagation:

- to see how the propagation functions, in the case of a correctly propagated field.

In this case you can read from top to bottom (which is the opposite direction of the propagation), or from bottom to top (in the direction the propagation proceeded).

- however, if you wish to know what caused a derailment of the propagation and ended at a field with nothing in common with what you wanted propagated:
  - **Do not:** Read from top to bottom by going up the propagation path, as this will only add to your questions about the propagation of the last and second last fields (when neither were supposed to be propagated).

- **Do:** Read from bottom to top, starting from one file field, and look for a point in which the propagation seems to have ceased processing the fields as expected.

You will quickly root out the problem on 1, 2 or 3 lines. Everything higher up is only a consequence of this derailment.

Once the problem has been located, consult the source of the program(s) concerned in order to help you understand what is contradicting the propagation principals.

## Going back up the propagation path


When you request a rollback through the propagation path while entering a program field, know that:

- a. It is better to read from bottom to top, to understand the path the propagation took as it was really carried out.
- b. TWO different interrogation methods are available:
  1. Screen 1 basic: go up the propagation while giving, for each field, another field which is the first cause of the field propagation.
  2. Screen 2 “by position” (accessible with **F10**): go up the propagation of each propagated position in a field, while giving, for each propagated position in a field, another propagated position in another field which is the first cause of the propagation for the field position.

The two screens often give exactly the same information, but they can vary significantly in the following case: One of the propagated fields is a complex field containing several propagated fields; in this case:

- Screen 1 can derive from one position to another.
- Screen 2 (by position) will give exact information.

To help you understand this nuance, here is a full example.

 **Example**

Going back over the propagation causes by field or by field position.

Propagation on the quantity fields.

File field `FIQTMIN` and `FIQTMAX`, Propa Letter: T

PG2: line `IF FIQTMIN >= QTMINI AND FIQTMAX <= QTMAXI ...`

Declaration of QTMINI and QTMAXI in a structure:

DSPARM DS


1 to 10 Article Code

11 to 16 `QTMINI` (lg 11,2) (packed)

17 to 2 `QTMAXI` (lg 11,2) (packed)

This DSPARM field is the unique entry parameter of program PG2.

PG1 calls PG2 with the DSPG2 field parameter:

 Declaration of `WQMINI` and `WQMAXI` in a structure:

DSPG2 DS

1 to 10 Code Article

11 to 16 `WQMINI` (lg 11,2) (packed)

17 to 22 `WQMAXI` (lg 11,2) (packed)

a. Going back over the propagation by Screen 1 (basic screen) for the `WQMAXI` of PG1:

- 1) `WQMAXI` of PG1P(11,2)                      Source line I DSPG2  
Field format: T10t
- 2) DSPG2 of PG1A(22)                            Source line PG2:DSPARM  
Field format: 10.(4-)(4-)/T10t/T10t/
- 3) DSPARM of PG2A(22)                        Source line I 11 16 QTMINI  
Field format: 10.(4-)(4-)/T10t/T10t/
- 4) QTMINI of PG2P(11,2)                      Source line IF FIQTMIN > QTMINI...  
Field format: T10t
- 5) FIQTMIN of PG2P(11,2)                    Source line File:XX, Fld:FIQTMIN  
Field format: T10t

If you read the information on each line separately, they are exact. It shows you the 1st propagation cause for each field. The same thing can be noticed for line 3: DSPARM was propagated first by QTMINI.

If, however, you search from bottom to top, it seems like `FIQTMIN` ends on `WQMAXI`! The cause is that DSPARM and DSPG2 contain 2 propagated positions.


b. Going back over the propagation by Screen 2 (by position) for `WQMAXI` of PG1 :

While specifying analysis of position 1,00

- 1) `WQMAXI` of PG1, pos 1,00P(11,2)Cause : DSPG2 of PG1, pos 17,01  
Field format: T10t
- 2) DSPG2 of PG1, pos 17,01A(22) Cause : DSPARM of PG2, pos 17,01  
Field format: 10.(4-)(4-)/T10t/T10t/
- 3) DSPARM of PG2, pos 17,01A(22) Cause : QTMAXI of PG2, pos 1,00  
Field format: 10.(4-)(4-)/T10t/T10t/
- 4) QTMAXI of PG2, pos 1,00P(11,2)Cause : FIQTMAX of PG2, pos 1,00  
Field format: T10t
- 5) FIQTMAX of PG2P(11,2)                      Cause : File:XX, Fld:FIQTMAX  
Field format: T10t

This time it is the real path the propagation followed. It shows you the 1st cause of propagation by field position, for every field position. In line 3: DSPARM position 17,01 was propagated first by QTMAXI.

Remember that the true reliable information is given by the interrogation by position.

 **Important!**  
The positions 17,01 (and not 17) concern the alphas containing the packed fields.

A small icon of a document with a right-pointing arrow, indicating a reference or tip.

**Reference**

For more information about field formats, refer to [Understanding field formats on page 44](#).

## 25 Setting propagation stoppers – ASETPRPSTP

The `ASETPRPSTP` command allows you to define the propagation stoppers. These will be stored in the list of propagation stoppers.

A stopper could concern:

- a program field (in this case fill in both cases).
- one field for all the programs (in this case, put a \* in program).
- a program for all the fields (in this case, put a \* in field).

### Note

If the name of the field for which you want to create a stopper is more than 10 characters, press F11 and you can continue the name below.

A maximum of 1000 stoppers of the type **\*EXCLUDE** can be defined in the list as well as 200 **\*VIAFLD** stoppers (there are no limits for **\*LDA** stoppers).

Even if you can modify the stoppers while the propagation is still running, those modified will only be taken into account at the next propagation execution.

### 1. **\*EXCLUDE**: Stop the propagation

This stops the propagation as soon as it reaches the field concerned (or at all the fields of a program).

Furthermore, if a propagated field is linked by a re-definition with a field for which an **\*EXCLUDE** propagation stopper was defined, then no propagation can proceed to the propagated field and on fields belonging to the re-definition. (Case of a DS in RPG or a de-composition of fields in COBOL.)

### Example

The re-definition of a multiple parameter text.

DS

```
1          50  PARAME
```

\* Table 1: Company title

```
1          30  LIBSOC
```

\* Table 2: Maxi Date for Order

```
1          6   DATMAX
```

The DATMAX field is a re-definition of PARAME but it is essential that there is no propagation between DATMAX and PARAME or DATMAX and LIBSOC.

Specify the PARAME type **\*EXCLUDE** as a propagation stopper. The DS will, therefore, not be modified.

**Note**

To avoid any propagation by the **\*LDA**, put the **\*LDA** field (and **\*LDAOCL36** for the OCL36) as a propagation stopper, type **\*EXCLUDE** for all the pgms (\*).

2. **\*VIAFLD**: Propagation via one or two intermediate variables

To be used in the case of fields used as parameters for a call to a routine (internal or external) and for which the returned value (in the same variable or in another) is then given in a variable result.

It is necessary to use a **\*VIAFLD** stopper when the intermediate field(s) are sometimes used to relay a value concerned by the propagation, and sometimes used in the same program to relay a value not concerned by the propagation.

If the called routine uses an input parameter and an output parameter, in this case indicate the second field in the **Associated field** column.

**Example**

The WSRVAL field is used to carry out a particular calculation.

\* Calculation based on an amount AMNT1

```
Z-ADDAMNT1          WSRVAL 154
EXSR   SRVAL
Z-ADDWSRVAL        WMNT1   72
```

\* or even (by a call of an external program)

```
CALL   ' PGVAL '
WMNT1  PARM   AMNT1  WSRVAL 154
```

\* Calculation based on a quantity AQTE

```
Z-ADDAQTE          WSRVAL
EXSR   SRVAL
Z-ADDWSRVAL        WQTE
```

The WSRVAL field is used as an input and output “parameter” for the calculation routine.

Specifying WSRVAL type **\*VIAFLD** without an associated field gives a correct propagation from the AMNT1 field to the WMNT1 field (or vice-versa). However, the WSRVAL field and the two AQTE and WQTE fields are not propagated.

**Example**

The WSRIN and WSROUT fields are used as input and output variables for a calculation routine.

\* Calculation based on an amount AMNT1

```

Z-ADDAMNT1      WSRIN  154
EXSR  SRVAL2
Z-ADDWSROUT     WMNT1   72

*or even (by a call from an external program)

CALL 'PGVAL2'
PARM AMNT1 WSRIN  154
WMNT1  PARM      WSROUT 154

* Calculation based on a quantity AQTE

Z-ADDAQTE WSRIN
EXSR  SRVAL2
Z-ADDWSROUT WQTE
    
```

The WSRIN field is used as an input parameter for the calculation routine. The WSROUT field is used as an output parameter.

Specifying WSRIN type \*VIAFLD with WSROUT as an associated field gives you a correct propagation between the fields AMNT1 and WMNT1 (or vice-versa). (However, the WSRIN, WSROUT fields and the two AQTE and WQTE fields are not propagated.)

**Note**  
The two above examples given in RPG are also valid in COBOL.

### 3. \*LDA: Re-organization of LDA positions

This time it is not a propagation stopper, but the possibility to configure a re-organization of the \*LDA following propagation (notably if you want to avoid the position modifications overtaking a certain position).

**Note**  
If you wish to put a real propagation stopper on the LDA, simply put an \*EXCLUDE type stopper with \*LDA or \*LDAOCL36 or even \*Lpppllll in the field name.

You indicate **PaaaaLbbbb** as a field name, where aaaa is the starting position in the \*LDA and bbbb is the corresponding length.

Possibly, you can indicate "PccccLdddd" as an associated field name where cccc is the new corresponding starting position and dddd is the new corresponding length.

Therefore, you can both divide the \*LDA into several independent parts (one overall one for the application and the other which can be redefined case by case for example), and also re-organize the position of these parts.

**Example**  
LDA defined in 2 independent parts to be inverted.



| Field      | Type | Associated field |
|------------|------|------------------|
| P0001L0060 | *LDA | P0601L0080       |
| P0061L0500 | *LDA | P0001L0600       |

The positions of the LDA situated between 1 and 60 are later found between 601 and 680. Those situated between 61 and 560 are later found between 1 and 600.

**Example**

LDA defined in 2 independent parts (so the position modifications for one should not affect the other).

| Field      | Type | Associated field |
|------------|------|------------------|
| P0001L0060 | *LDA | P0001L0060       |
| P0061L0500 | *LDA | P0061L0500       |

The positions of the LDA situated between 61 and 500 will not be affected by the modifications made between positions 1 and 60.

**Note**

There is no size checking for the defined parts; it's up to you to build the non-overlapping parts for which the new size is greater than or equal to the old one.

## 25.1 Consulting literals detected by the propagation

The propagation detects the literals (numeric or alphanumeric) which are sometimes used to load a value in a field, or to compare a field in relation to the values directly in the program.

In `AWRKFRMCHG` and see new source, you can see that the propagated literals are flagged by underlining.

However, it can be useful to know the complete list of different literals flagged by the propagation, and that being for any program.

- No interrogation dedicated to that purpose exists.
- But, you can get this information anyway, via SQL or QUERY:
  - on the ARCAD database file that contains the line flags: `ARDCHXF1`.
  - by limiting yourself to the started propagation (name and library)
  - by requesting the literal flags.

```
==> SELECT CXDRVL, CXDRVF, CXAPP, CXVER, CXPGMS, CXTYPS, CXFL1, CXFLC1,
CXFLD
FROM ARCAD_PRD/ARDCHXF1
WHERE CXDRVL = 'list file library of derived fields' (Version library)
AND CXDRVF = 'list file name of derived fields' ('LSTDRVFLD')
AND CXFLTY = 'L' (Literals)
```

This will give you the:

- **CXAPP**: The application code
- **CXVER**: The version N° of cross-references
- **CXPGMS**: The source program name
- **CXTYPS**: The source program type
- **CXFLL1**: The source line n° (7 characters)
- **CXFCL1**: The column n° in the line
- **CXFELD**: The value of the propagated literal (max 30 characters).

**Note**

Despite the existing option in `ACVTPGMFLD` (Conversion of literals), there are currently no automatic modifications of literals in ARCAD Transformer Field.

## 26 Applying generated modifications – AAPYFRMCHG

The `AAPYFRMCHG` command allows you to apply the modification propositions, previously generated and visible by `AWRKFRMCHG`. These modifications will have been checked and validated by the developer. This option creates new sources in the version library for each item needing modification.

You should execute this command in BATCH as it is relatively long. After command execution, the modified items will be accessible with the command **Working with objects/members** – `AWRKOBJARC`.

The following flagging will be placed at the beginning of the source lines, in position 1 to 5 in RPG, RPGLE, COBOL and in comment `/* ... */` in CLP:

- **Position 1 to 3:** The modification flagging characters of the FLAG parameter. The 1st character will be replaced by a W if the line is in Warning.
- **Position 4:** A character from the COLOUR attribute: See Configuration of flagging by FRM.
- **Position 5:** A character visualizing the type of modification:
  - V: Verified Line (Just flagged)
  - C: Changed Line (Modified)
  - I: Inserted Line (New line)
  - D: Deleted Line (if shown in comment)

The following table presents the different parameters of the `AAPYFRMCHG` command.

| Parameter                            | Description   |
|--------------------------------------|---|
| Environment (ENV)                    | You can specify the environment in which you wish to run the command.   |
| List of modified components (TOLIST) | You must specify the name of the list that will receive the names of the modified items. This output list from the process, can then be used for different operations.  |
| Replace or add record (OPTION)       | Specify if the list file defined in the TOLIST parameter must be replaced if it already exists.<br>*REPLACE: The list file will be reinitialized at the beginning of process.<br>*ADD: The file is not reinitialized; only the new items will be added. |

Table 32: AAPYFRMCHG command parameters

| Parameter                       | Description  |
|---------------------------------|--|
| Flagging character (FLAG)       | <p>The flagging character will be present on all the source lines generated or automatically modified.</p> <p>*INTNBR: The flagging character is chosen by the system. It is composed as follows: M + Internal chrono N°</p> <p>Alpha Value: This value must not begin with two stars.</p> <p>Avoid also the <b>W</b> in the 1st character as the lines in warning will have a <b>W</b> as the first character which will overwrite your first flagged character.</p>  |
| Conserve old lines (CSVOLDLIN)  | <p>This allows you to indicate if you want to keep the previous line (in comment) in the case of a line modification or deletion.</p> <p>*YES: In the new source, you will first have the old line (in comment) then the new line (except in the case of a line deletion).</p> <p>*NO: In the new source, you will only find the new line (or nothing at all in the case of a line deletion).</p>  |
| Reference document (REFDOC)     | <p>It is possible to specify from this command, to which user document you want to apply the automatic modifications (you must first create the document and note its number).</p> <p>If you leave the default value *NONE, this information will be required when you use one of the source editors.</p>  |
| Source backup (SAVE)            | <p>This allows you to indicate if you wish to back up your sources before transformation.</p> <p>*YES: The sources will be copied in the QTMP SRC file of the backup library.</p> <p>If the source already exists in this library, the name will be shortened using numbers in order.</p> <p>*NO: The sources will not be copied.</p> <div style="border: 1px dashed gray; padding: 5px; margin-top: 10px;"> <p><b>Note</b></p> <p>This option is only useful when your version already contains the modified items before the command execution.</p> </div> |
| Update transfer log (UPDTFRLOG) | <p>This parameter allows you to update the sources transfer history log.</p>   |

Table 32: AAPYFRMCHG command parameters

## 26.1 Compiling the modified and dependent objects

### Important!

A macro-command is given here for starting the linked compilation of modified components and their dependencies (components not modified)



but to be re-compiled).

This `ACPLOBJDEP` macro-command must only be used in the case of a simple modification where you won't need to verify the modified sources beforehand. Furthermore, if a compilation incident intervenes (for example on a file), the programs will be re-compiled with the previous file version.

It is preferable to proceed *less* automatically by the command `AWRKOBJARC`: Work with objects/members.

1. Verify and compile the reference file(s), if there are any.
2. Select the PF\*, verify and compile them. The logical files will automatically be compiled.
3. Verify and compile the PRTF/DSPF.
4. Verify and compile the modified programs.
5. With F19, run the re-compilation of other components not modified but those that need re-compiling.



#### **Reference**

For more information about these operations, refer to the ARCAD Skipper documentation.

# 27 Working with other options in the TRANSFORM menu

---

## Chapter Summary

|  |     |
|--|-----|
| 27.1 Configuring transformer marking parameters.....                 | 126 |
| 27.2 Deleting information generated by propagation – ACLRDVRFLD..... | 127 |
| 27.3 Converting a list of fields to a list of sources – ACVTLST..... | 127 |

The TRANSFORM menu has other significant options that have been covered in this section.

## 27.1 Configuring transformer marking parameters

---

This configuration should be verified before applying the modifications (`AAPYFRMCHG`).

It allows you to set the colors of the source lines flagged and/or modified by ARCAD Transformer Field.

Added lines imply:

- Actual added lines
- Verified lines
- Modified lines

Deleted lines imply:

- Actual deleted lines (in comment)
- The previous version of a line (now in comment)

The indicated value corresponds to the hexadecimal code which is transformed into a color attribute.

Modif. of flagging parameters

Modified line presentation attributes:

|                        |    |              |
|------------------------|----|--------------|
| Deleted lines. . . . . | 28 | 21= xxx      |
| Added lines. . . . .   | 22 | 22= xxx      |
|                        |    | 23= xxx      |
|                        |    | 28= xxx      |
|                        |    | 29= xxx      |
|                        |    | 30= xxx      |
|                        |    | 31= xxx      |
|                        |    | 32= xxx      |
|                        |    | 33= xxx      |
|                        |    | 37= xxx      |
|                        |    | 38= xxx      |
|                        |    | 40= No color |

## 27.2 Deleting information generated by propagation – ACLRDVFLD

---

The `ACLRDVFLD` command can be considered as a purge command that allows you to eliminate information from the ARCAD files generated by the propagation.

Its execution is not obligatory but is recommended at the end of processing a problem using ARCAD Transformer Field, when you no longer need to consult the propagation information.

When you run this command, only one parameter is required:

- The name and library of the derived fields list file (LSTDRVFLD)

The process deletes:

- The propagation information stored in the files in the ARCAD database (ARD...F1).
- Then the list will be deleted.

 **Warning!**

This purging does not remove the source line flags or modifications stored in ARDCHSF1, visible by `AWRKFRMCHG`.

There is no need to run this command when you restart an entire propagation. It is automatically run by the propagation in Step 1 – Entry in the programs.

## 27.3 Converting a list of fields to a list of sources – ACVTLST

---

In the FRM menu, you can find an option allowing to run the `ACVTLST` command for various conversion types like the options `*FLDTOMBR`, `*OBJTOMBR` and `*MBRTOOBJ`.

You must specify a new output (type M = source member).

It allows you to obtain a list of source members from a detailed list of fields of these source members.

You can use this option at any time, for example:

- To obtain the list of physical files from the list of fields to propagate.
- To obtain the list of impacted components (Program and LF) after having started steps 1 and 2 of the propagation (from the LSTDRVFLD).
- To obtain the list of display fields and impacted printouts (DSPF and PRTF) after having started steps 1 and 2 of the propagation (from the LSTDDSFLD). You will also be given the PF and LF in this list.



# DATA RECOVERY



## 28 Recovering data

---

### Chapter Summary

|  |     |
|--|-----|
| 28.1 Working with data recovery sequences.....             | 129 |
| 28.2 Working with associated files for data recovery.....  | 132 |
| 28.3 Generating data formatting programs – AGENFMTPGM..... | 135 |
| 28.4 Executing data formatting programs – AEXCFMTPGM.....  | 139 |

**Note**

The data formatting or data recovery concepts are identical for all of the products in the ARCAD-Transformer suite.

The data recovery managed in ARCAD Transformer Field has the objective to allow you to carry out the automatic execution of a process from a list of physical file fields. The process will be the same (or nearly the same) for each field on the list.

The process is executed by the programs generated by ARCAD (one program per file). This program declares the file as primary and in update (or as read only, if requested). Each file record is, therefore, read once.

The process to carry out for each field is found in the main part of the program. Each record is updated at the end of the process (if the file was open in update). Unlike other transformer processes, it is possible to use this command from a list of fields of physical files without source.

The data formatting commands can be used to:

- fill in the extended part of fields (following an extension of a file's fields). The previous data having already been copied (in \*MAP \*DROP, aligned on the left for an alphanumeric field, aligned on the point for a numeric field).
- build the new fields (following a field addition in the files). The old data having already been copied (in \*MAP \*DROP).
- carry out an update process (conversion) for the value of certain fields (In this case, there is no modification of the file structure).
- carry out an update process (conversion) for the values of certain fields while keeping a trace, in an associated file, of the old field value (or the conversion difference).
- automate the starting of a process which must be run for each one of the values in a complete series of physical file fields.
- etc...

Using a standard routine written in RPGLE, you can describe (for one copy only) the process to carry out. It is also possible to create an associated file for each file processed.

### 28.1 Working with data recovery sequences

---

The following two commands allow data recovery:

- Generation of PF data field formatting - `AGENFMTPGM`
- Execution of data formatting programs for PF fields - `AEXCFMTPGM`

These two commands are executed at different times in different environments. It is also necessary to pre-configure them.

### 28.1.1 Write a standard routine

Using the `AWRKSTDRTN` command, you must write (or check the existence of) a standard routine in RPGLE and only containing C cards.

This routine contains the part of the source corresponding to the process to execute for each file field.

In this routine, you can use the FRM (\$\$) variable names which will be substituted by the variable names of your files (and also the field length, nb. of decimals, field format, etc...).

You can also anticipate the different parts of source code according to certain cases, by testing with the Routine Check language that lead to the insertion of such and such blocks of code contained in the routine.

If you have already generated recovery programs from a routine and find that the generated code does not respond to your needs, you can modify the routine again; you must then re-generate the recovery programs.

### 28.1.2 Verify or update the impacted fields list

The generation and execution of recovery programs is based on a list of physical file fields to process. This list can:

- already have been used by the propagation without modification.
- already have been used by the propagation with modification of the PF sources.
- be specifically created for generating recovery programs, without going through propagation.

#### Reference

For more information on the physical file fields, see [Working with field lists on page 59](#).

#### Important!

You can copy the list, modify its contents (notably `LST_CSST` and `LST_JZSEL2`) before using it for generating recovery programs.

Do not hesitate to modify the `LST_CSST` and `LST_JZSEL2` fields, if necessary, to put either other file fields or even the values which will be tested in the standard routine in order to insert a block of code according to the case.

In the list of fields, the fields given in the below table will be used to generate the recovery programs.

| Field       | Length/Type | Variable                           | Contents  |
|-------------|-------------|------------------------------------|---|
| LST_ JOBJ   | 10A         | \$\$F1                             | File field to process   |
| LST_ CSST   | 10A         | \$\$F2                             | New file field  |
| LST_ JZSEL2 | 30A         | \$\$F3                             | Third file field  |
| LST_ CTYPE  | 10A         | \$\$TY,\$\$LG,\$\$LD               | Field type, length and Nb. of decimals  |
| LST_ CCPLT  | 7A          | \$\$FM                             | Field format (in transformer terms)   |
| LST_ CTXT   | 50A         | \$\$TX                             | Text of complete field  |
| LST_ CTXT   | 5x10A       | \$\$T1,\$\$T2,\$\$T3,\$\$T4,\$\$T5 | Each of the 5 parts of the text field, with 10 characters (from 1 to 10, 11 to 20, etc....) |
| LST_ JLIB   | 10A         | \$\$DM                             | File record format  |
| LST_ JSRCF  | 10A         | \$\$DF                             | File name   |
| LST_ CATR   | 10A         | \$\$DT                             | Type of file (PF,PF38 or PF36)  |
| LST_ JXLIB  | 10A         |                                    | File library (for FMTFLDDTAX with *FROMLIST)  |
| LST_ JZSEL1 | 10A         |                                    | Receives the name of the recovery program generated by file.                                |

Table 33: Fields used to generate recovery programs

### 28.1.3 Generate data field formatting programs – AGENFMTPGM

Objective: To automate the writing of similar programs processing the file fields.

This command is run in a development environment (ARCAD or possibly one not managed by ARCAD).

It generates program sources (and objects) dedicated to the task defined in the standard routine.

**Important!**

If the files to be processed undergo modification (extension or addition of fields) via ARCAD Transformer Field, you must wait until the new files are created (application of modifications + compilation) before starting the generation and compilation of these recovery programs. Otherwise, the compilations will be carried out according to the previous file description.

#### Correction – Personalization of generated programs

If you wish, you can modify the generated programs and recompile them afterwards.

**Reference**

For more information about AGENFMTPGM, refer to [Generating data formatting programs – AGENFMTPGM on page 135](#)

## 28.1.4 Execute data formatting programs – AEXCFMTPGM

Objective: To automate the execution of recovery programs, file by file, member by member.

This command can be run:

- In the version where the recovery programs are generated.
- In the test environment.
- In the production environment.
- With an online list of libraries, chosen at the start up.
- On the AS400 where the programs are generated.
- On another AS400 if the generated programs were distributed.

**Note**

This command does not require an ARCAD Transformer Field license; you just need to have installed ARCAD with an ARCAD Process Manager license.

It is necessary to have the same list of fields for executing the process as for generating the recovery programs.

At the beginning of the process, there is a quick verification for the presence of recovery programs (and possibly any associated files); any anomaly will stop the process from beginning.

In case of failure at the program's execution on a file, the process continues on to the next part; the non-processed file fields change to status **Abn**. Once the problem is corrected, it is possible to restart the process in the list while specifying **Restart after abnormal end = \*YES**.

**Reference**

For more information about AEXCFMTPGM, refer to [Executing data formatting programs – AEXCFMTPGM on page 139](#).

## 28.2 Working with associated files for data recovery

Objective: To store information coming from the processed file.

For each record processed, a record can be created in an associated file.

Each field of the processed file has a corresponding field (with the same name) in the associated file. Furthermore, the associated file also includes the primary keys of the processed file in its description. The other fields of the processed file are not present in the associated file.

Utilization cases of an associated file:

- To keep an old value of fields before conversion.
- To trace the records which will be detected by the recovery program process.
- To show the recovery program's use on the records of the processed file.
- etc....

These associated files are not generally used directly in the application. They can be consulted via SQL, Query or by program.

### 28.2.1 Generate an associated file

The associated file has the same name as the processed file, but its description (source and object) is placed in a different library (associated file description library).

Therefore, there are as many associated files as there are processed files.

Its source is generated which contains the following fields:

- Primary keys of the processed file (same name, type and length as in the processed file; the field name is preceded with a K if the key field is also processed).
- Processed fields (same name as the field of the processed file (`LST_JOB`), same text. The type, length and number of decimals can be identical to the field of the processed file or the same everywhere).

### 28.2.2 Execute recovery program with an associated file

To use the associated files, you must specify two library names when running the `AEXCFMTPGM` command:

- Associated file description library: here you will find the empty file object generated by the associated file.
- Associated file data library: the empty associated files will be created in this library, then filled by the process of recovery programs.

These two libraries can be the same; however, you should verify that the associated file data library corresponds to the environment for which you ran the process. There are normally as many associated file data libraries as there are different executions of recovery programs.

#### Note

Never put these two libraries online – there would be a risk of confusion between the file to process and the associated file (description or data).

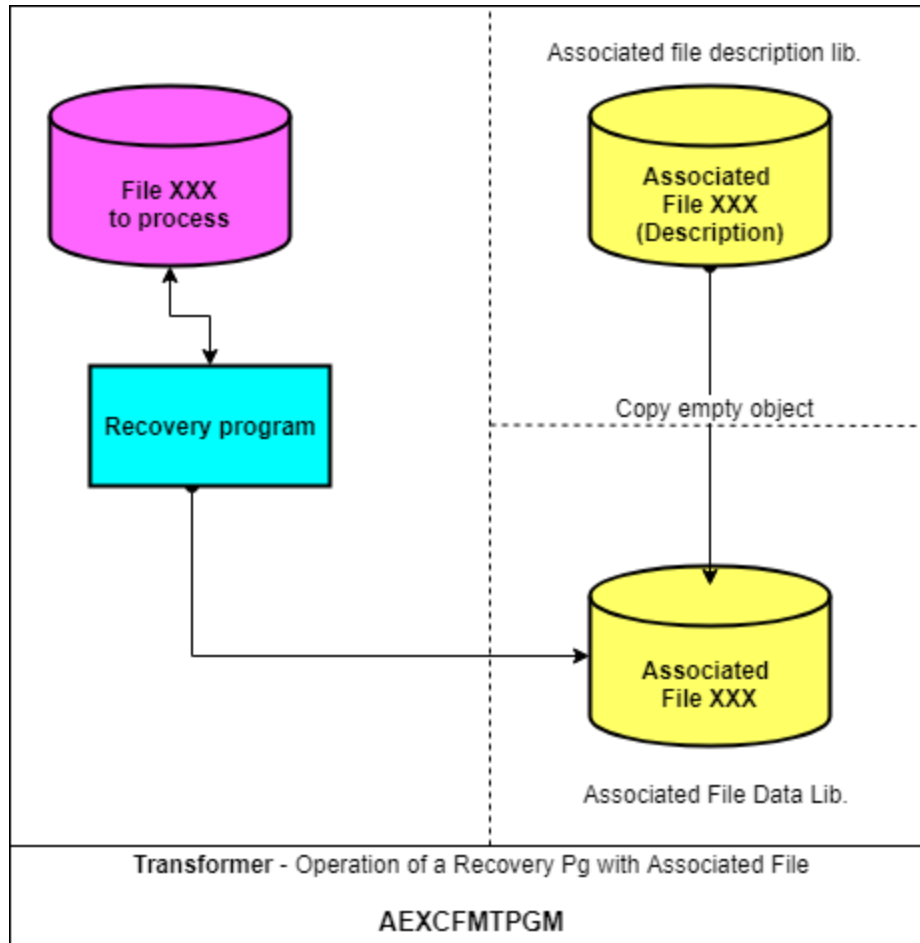


Figure 9: Associated File Data Library

### 28.2.3 Specify contents of standard routines for associated files

In this case, a new FRM variable is available:

\$\$A1 - Indicates the field of the associated file which has the same name in the PF, but which is prefixed by F12\_ in the generated RPGLE program.

Furthermore, to create the record in the associated file, insert the following process to the standard routine:

```
C           Eval      CreateAssF = '*YES'
```


The record creation is only made after the processing of all the fields and only if the **CreateAssF** variable is at \*YES.

Also, before writing, the keys of the associated file are loaded from the corresponding fields in the processed file.

#### Description of primary keys

To obtain a complete associated file, you must know the list of file fields which are considered as primary keys.

The primary keys correspond to the list of file fields which are considered to be the only identifier of the file's record.

 **Example**

Command N° + Line N° for a command line file.

These primary keys are loaded after running the command **Create database links** - `AUPDFLDDBR`. In this loading process, the following are considered as primary keys:

- the PF keys if they exist with the key word UNIQUE.
- if these are absent, then the keys of the first LF with the key word UNIQUE (by looking at the file xxxL1, xxxL2, then xxxLA, xxxLB, then the others).
- if also absent, then the PF keys if they exist without the keyword UNIQUE.
- if also absent, then the keys of the first LF without the key word UNIQUE (by looking at the file xxxL1, xxxL2, then xxxLA, xxxLB, then the others).
- if also absent: No primary key is found.

The method for finding the primary keys does not necessarily correspond to the way your physical and logical files were conceived; it is necessary to verify all the primary keys of all the physical files by:

- `ADSPFLDDCT` – Field Database Repository
- Option 35 `DSPPFKEY` – Display the Primary Keys
- Option 32 `WRKPFKEY` – Work with the Primary Keys

Modify the primary keys if they don't correspond to what was expected. Your modifications will be retained even if you restart the `AUPDFLDDBR` command.

## 28.3 Generating data formatting programs – AGENFMTPGM

The `AGENFMTPGM` command allows you to generate programs for formatting the data of the physical file fields from the database. By formatting, we mean all the processing to be carried out from the PF fields, whether its updates, check processes or even program calls to run for each field.

You can also automate the creation of recordings in an associated file which is also generated.

Once generated, these programs can be executed by the `AEXCFMTPGM` command based on the same list.

For the generation of programs, the physical files should have already been created in a new format if you want to run the compilation of these recovery programs.

The source programs (and object, if the `PGMCPL` parameter is at \*YES) are generated in the library indicated at the `PGMLIB` parameter.

These generated programs are named according to the `PGMNAM` and `PGMNUM` parameters (for example: `FMTF0001`, `FMTF0002` etc...).

A program is generated for each file present in the field list.

**Important!**  
Run this process in batch to avoid answering questions concerning the destination library, of the source and object, for each program (as can be the case interactively!).

The following table presents the different parameters of the `AGENFMTPGM` command.

| Parameter  | Description   |
|--|---|
| List of impacted fields (LIST)                   | Enter the name of the list file library that contains the database fields for which you wish to carry out the field formatting.   |
| Sub-routine for process (STDRTN)                 | Specify here, the name of the standard code routine which will be inserted into the data formatting programs for each field to be processed. To work with routines, use the <code>AWRKSTDRTN</code> command.<br><br>This routine must be written in RPGLE.  |
| Propagation group (PRPGRP)                       | *ANY: All the files present in the fields list are processed (whatever the contents of the field <code>LST_CCPLT</code> – Field format are).<br><br>Propagation group + Identification letter : Only those fields with a field format including the specified identification letter are processed.  |
| Work with associated files (ASSFILE)             | Specify if you want to work with an associated file for each file processed.<br><br>*NO: No associated file is generated.<br><br>*YES: An associated file is generated for each file processed. In this case you should fill in the parameters concerning the associated file.  |
| Associated File-Description Library (ASSFDSCLIB) | This parameter is obligatory when you want to work with an associated file.<br><br>Enter the name of a library which is not part of your application; if the indicated name doesn't exist, it will be created automatically.<br><br>In this library, the <code>AGENFMTPGM</code> command creates the source of the new associated files having the same name as each processed file. This file will contain, for every processed field, a field having the same name and text but for which the description (Type and Length) can be different. Furthermore, this file will also contain the primary keys of the processed file which allows the correspondence between the original file and the associated file. (The definition of these primary keys is accessible with the command <code>AWRKPFKEY</code> .) |
| Associated file-Text (ASSFTEXT)                  | Obligatory parameter for generating an associated file.<br><br>Specify the text that will be given to the associated files generated; the name of each processed file will be placed at the end of this text.   |
| Associated file-Indexed (ASSFKEYED)              | Obligatory parameter for generating an associated file.<br><br>*NO: The associated file generated is not indexed on the key fields.<br><br>*YES: The associated file generated is indexed on the key fields.  |

Table 34: `AGENFMTPGM` command parameters





| Parameter                                | Description  |
|--|--|
| Associated file-Field type (ASSFLDTYPE)  | <p>Obligatory parameter for generating an associated file.</p> <p>Specifies the type, that every processed field in the associated file, will have (except the keys).</p> <p>*SAME: The field type will be the same as the one in the original file.</p> <p>Type: Enter the type of the PF field which will be valid for all the fields of the associated file.</p>  |
| Associated file-Field length (ASSFLDLEN) | <p>Obligatory parameter for generating an associated file.</p> <p>Specifies the length that every processed field in the associated field will have (except the keys).</p> <p>*SAME. The length of the field will be the same as the one in the original file.</p> <p>Length: The indicated length will be valid for all the fields of the associated file.</p>  |
| Associated file-Field dec.# (ASSFLDDEC)  | <p>Obligatory parameter for generating an associated file.</p> <p>Specifies the number of decimals for every field processed in the associated file (except the keys).</p> <p>*SAME: The number of field decimals will be the same as the one in the original file.</p> <p>Length: The number of indicated decimals will be valid for all the fields of the associated file.</p>   |
| Name of the programs created (PGMNAME)   | <p>This is a model of the name of the recovery programs which will be generated.</p> <p>Three or four consecutive dots must be in this model. These will be replaced by a chronological number for each program to generate.</p> <div style="border: 1px dashed green; padding: 5px; margin-top: 10px;"> <p> <b>Example</b><br/>PG...R1 or PGM....</p> </div> |
| First program number (PGMNBR)            | <p>This is the starting value for the chronological number allowing you to create the generated program names.</p> <div style="border: 1px dashed green; padding: 5px; margin-top: 10px;"> <p> <b>Example</b><br/>With the 'PG...R1' model and the starting N° 251, the programs will be named PG251R1, PG252R1, etc...</p> </div>                            |
| Generated program text (PGMTEXT)         | <p>Enter the text that will be given to the generated programs; the name of each processed file will be placed at the end of this text.</p>  |

Table 34: AGENFMTPGM command parameters

| Parameter                           | Description  |
|-------------------------------------|--|
| Recovery programs library (PGMLIB)  | <p>This is the library name that will receive the sources and objects of the recovery programs.</p> <p>The value *CURENV corresponds to the current version library.</p> <div data-bbox="501 394 1297 510" style="border: 1px dashed gray; padding: 5px;"> <p><b>Note</b><br/>This library does not have to be managed by ARCAD. If it doesn't exist, it will be created.</p> </div>   |
| Target release (TGTRLS)             | <p>Indicates under which OS edition the created object must be processed.</p> <div data-bbox="501 583 1297 678" style="border: 1px dashed gray; padding: 5px;"> <p><b>Note</b><br/>This parameter only applies to generated programs.</p> </div> <p>The possible values are as follows:</p> <p>*DFT: The <code>ACPL</code> command will use the default value which was applied to the corresponding compilation command (<code>CRTBNDRPG</code>).</p> <p>V3R2M0: Indicating this system version leads to restrictions on the lengths of fields in generated RPGLE programs: the lengths of field names are limited to 10 characters; if you generate an associated file, the names of RPGLE fields of the associated file are renamed in the I card.</p> <div data-bbox="501 989 1297 1104" style="border: 1px dashed gray; padding: 5px;"> <p><b>Reference</b><br/>For more information on the other possible values, refer to the IBM documentation.</p> </div> |
| Compile generated programs (PGMCPL) | <p>This parameter indicates if you should compile (or not) the generated recovery programs. You can compile them later using option 9 in the <code>AWRKOBJARC</code> command.</p>  |
| File is being updated (UPDATE)      | <p>Indicates if the generated process must open the file being updated and carry out an UPDATE for every read record (after processing).</p> <p>*YES: The file is opened in update in the recovery program. Each read record is therefore locked; it is updated once again after the processes are carried out on each field.</p> <p>*NO: The file is opened in read-only in the recovery program. There won't be any locking of records or updates. This is useful for processes that must not modify the contents of fields in the file; however you can work with an associated file (for which records are being added to) in this case.</p>   |

Table 34: AGENFMTPGM command parameters

| Parameter                               | Description  |
|---|--|
| Reference document (REFDOC)             | It is possible to specify from this command, which user document you want the generated programs attached to. If you leave the default value *NONE, you will be asked for this information once you enter one of the source editors (if you generate your programs to a version managed by ARCAD).               |
| Restart after abnormal end? (RUNAFTABN) | Allows recoveries in the case where the command was not run on all the required items. <div style="border: 1px dashed gray; padding: 5px; margin-top: 10px;"> <p><b>Note</b><br/>You can check the non-processed items by displaying the list (command <code>AEDTLST</code> or <code>ADSPLST</code>).</p> </div> |

Table 34: AGENFMTPGM command parameters

## 28.4 Executing data formatting programs – AEXCFMTPGM

The `AEXCFMTPGM` command allows you to link up the execution of programs allowing the formatting of database physical file fields. The execution of recovery programs will take place file by file, data member by data member. By formatting we mean all the process to be carried out from the PF fields, whether updates, verification processes or even program calls to execute for each field.

This command can be used on a remote machine that only has ARCAD Process Manager (without ARCAD Transformer Field).

The programs to run were generated by the `AGENFMTPGM` command (or even created or modified by the user). Their names are in the list of PF fields (one program per file) in the `LST_JZSEL1` field (selection field).

This process first checks (before any execution) that all the necessary programs exist as objects (and also the descriptive objects of associated files, if requested).

**Important!**  
When this is done after a database modification, the process should only be run once the data has been retrieved from the new files (with the option `*MAP *DROP`).

**Note**  
In the case of non-database files (Type PF36), the generation command (`AGENFMTPGM`) can be run at any time.

However, for the execution by `AEXCFMTPGM` it will be necessary to verify beforehand (and update) the actual name of each file (and possibly its library too) in the list of fields to process, as the name memorized in the list may contain “Joker” characters instead of their real value.

In this case, it is recommended to copy the list of fields in order to keep the original too (by a `CRTDUPOBJ` or an `AEXTLST` but not by a `CPYF`).

The following table presents the different parameters of the `AEXCFMTPGM` command.

| Parameter  | Description  |
|--|--|
| List of impacted fields (LIST)                   | Indicates the name and library of the list file containing the database fields for which you wish to carry out field formatting.   |
| Database file library (DTALIB)                   | <p>The DTALIB parameter allows you to indicate in which library, the physical files to process, are found.</p> <p>There are 2 possible values for this parameter:</p> <p>*LIBL: The physical files are found in one of the libraries of the job libraries list. (Recommended option as long you have correctly verified the libraries list.)</p> <p>*FROMLIST: The name of the library containing each physical file is the one present in the Fields List file (indicated in the LIST parameter).</p> |
| Identification letter (PRPLET)                   | <p>*ANY: All the fields on the fields list are processed.</p> <p>Identification letter: Only the fields, having a field format with the specified identification letter, will be processed. There is no existence check for this letter in a propagation group.</p>  |
| Work with associated files (ASSFILE)             | <p>Specify if you wish to work with an associated file for each processed file.</p> <p>*NO: No associated file is given.</p> <p>*YES: Each associated file will be created and filled in relation to its description.</p> <p>Of course, you must have planned this associated file directly after the command for generating formatting programs (AGENFMTPGM). In this case you must fill in the parameters concerning the associated file.</p>  |
| Associated file-description library (ASSFDSCLIB) | <p>This parameter is obligatory when you want to work with an associated file.</p> <p>When executing, you must re-indicate the library name, already indicated in AGENFMTPGM, so that AEXCFMTPGM can find the descriptions of associated files.</p>  |

Table 35: AEXCFMTPGM command parameters


| Parameter                                | Description   |
|--|---|
| Associated file-data library (ASSFDALIB) | <p>This parameter is obligatory when you want to work with an associated file.</p> <p>When running recovery programs, this library will regroup the data of associated files.</p> <p>The associated files in this library will be cleared before executing the recovery programs if they are already present; however, if they don't exist, they will be created using the object present in the descriptive library of associated files.</p> <p>You can indicate the same name as the descriptive library of associated files or a different name, at each execution. The library will be created at execution if it doesn't exist.</p>  |
| Restart program library (PGMLIB)         | <p>This parameter is obligatory.</p> <p>This is the name of the library that contains the recovery program objects.</p> <p>The *CURENV value corresponds to the current object library.</p> <p>The *LIBL value indicates that the recovery programs will be searched for in the libraries in the job LIBL.</p>  |
| Output library non DB file (OUTLIB)      | <p>If not given, you must indicate the name of a library that will receive the formatted files.</p> <p>If this library doesn't exist, it will be created.</p> <p>The new formatted files are created without a DB description, in the form of one single field for the entire record.</p> <p>Once the <code>AEXCFMTPGM</code> process has finished, you must delete the old files (and their eventual index) in the original library, then re-create them taking into account the new description (same for the eventual index) and then re-copy the data from the OUTLIB library by a CPYF at LVLCHK(*NO).</p> <div data-bbox="509 1188 1297 1377" style="border: 1px dashed orange; padding: 10px; margin: 10px 0;"> <p> <b>Important!</b></p> <p>This parameter is only necessary for the execution of data formatting for non-database files, that is, those having the attribute "PF36" in the fields list. If there are none in your fields list, leave this field empty.</p> </div> |
| Recovery after abnormal end? (RUNAFTABN) | <p>Allows recoveries in the case where the command was not run on all the items.</p> <p>You can check the non-processed items by displaying the list (<code>AEDTLST</code> or <code>ADSPLST</code>).</p> <p>*NO: No recovery after abnormal end. If you re-run the command, it will be re-executed on all selected items.</p> <p>*YES: A recovery is carried out. The command will be applied to all items on the list not having the <b>Ok</b> status.</p>   |

Table 35: AEXCFMTPGM command parameters

## 28.4.1 Set Example 1 – Data recovery

Objective: To write recovery programs that allow you to carry out a calculation (default currency -> alternate currency) for each file field; the results are placed in the new file field. The conversion difference is placed in the 3rd file field.

### The routine for this process

```
C*%CM=====
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C*   Calcul. DIFFERENCE field and file field allocations before file UPDATE
C      CALL      'CVZECRFI'
C      PARM      $$F1          WCVTIN          30 9
C      $$F2      PARM      $$F2          WCVTAL          30 9
C      $$F3      PARM      PARM          WCVTEC          30 9
C      PARM      $$LD          WCVTDC          2 0
C*%EXIT
```

### Contents of the Field list to process

| File     | Origin | Field  | Utilization | Field  | Selection 2 | Type                         |
|----------|--------|--------|-------------|--------|-------------|------------------------------|
| CLIENTP1 | CLIASS | CLIAAS | CLIAAS      | CLIAES | P(10,0)     | Amt diff. Euro to Fr. Insur. |
| CLIENTP1 | CLICAF | CLIAAF | CLIAAF      | CLIAEF | P(15,2)     | Amt diff. Euro to Fr. T.O.   |

### Generated recovery program

```
FCLIENTP1 UP E DISK
C*
C*****
C* Field CLIASS : Amt diff. Euro to Fr.
C* Format T Type P(10,0)
C*
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C* Calcul. DIFFERENCE field and file field allocations before file UPDATE.
C      CALL      'CVZECRFI'
C      PARM      CLIASS          WCVTIN          30 9
C      CLIAAS    PARM      CLIAAS          WCVTAL          30 9
C      CLIAES    PARM      PARM          WCVTEC          30 9
C      PARM      0              WCVTDC          2 0
C*****
C* Zone CLICAF : Amt diff. Euro to Fr. T.O.
C* Format T Type P(15,2)
C*
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C* Calcul. DIFFERENCE field and file field allocations before file UPDATE.
C      CALL      'CVZECRFI'
C      PARM      CLICAF          WCVTIN          30 9
C      CLIAAF    PARM      CLIAAF          WCVTAL          30 9
C      CLIAEF    PARM      PARM          WCVTEC          30 9
C      PARM      2              WCVTDC          2 0
C*****
```

The called program (CVZECRFI) is given the task of carrying out the conversion.

## 28.4.2 Set Example 2 (with associated file) – Data recovery

Objective: To carry out a conversion of “Franc -> Euro” in a file with a non-modified structure, while keeping the traces of conversion differences in an associated file (in a packed field of 5, including 4 decimals).

### The routine for this process

```
C*%CM=====*/
C*%CM  Routine calculation for Big Bang, with associated file      */
C*%CM=====
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C*      Calculation amount in Euro, differences stored in associated file.
C          CALL      'CVZECRFI '
C          PARM      $$$F1          WCVTIN          30 9
C      $$$F1      PARM      0          WCVTAL          30 9
C          PARM      WCVTEC          30 9
C          PARM      $$LD          WCVTDC          2 0
C          Z-ADD     WCVTEC          $$A1
C          Eval      CreateAssF = '*YES'
C*%EXIT
```

### Contents of the Field list to process

| File | Origin   | Field  | Utilization | Field  | Selection 2 | Type    | Text                        |
|------|----------|--------|-------------|--------|-------------|---------|-----------------------------|
|      | CLIENTP1 | CLIAAS | CLIAAS      | CLIAES |             | P(10,0) | Amt dif. Euro in fr. Insur. |
|      | CLIENTP1 | CLICAF | CLIAAF      | CLIAEF |             | P(15,2) | Amt dif. Euro in T.O.       |

### Primary keys of the File

- CLICOD -> Client Code
- CLIASS -> Insurance amount (show the particularity of also being in the list of fields to process). This field is only keyed for the example as generally the only identifier for the Client file is the Client Code!

### Generated associated file

```
A          R CLIENTF1
A* Primary Keys
A          CLICOD          8A          TEXT('Code Client          ')
A          KCLIASS        10P 0      TEXT('Insurance amount      ')
A* Fields
A          CLIASS          5P 4      TEXT('Insurance amount      ')
A          CLICAF          5P 4      TEXT('T.O. amount           ')
A* Keys
A          K CLICOD
A          K KCLIASS
```

### Generated recovery program

```
FCLIENTP1 UP E          DISK
F* %EXEC OVRDBF FILE(ASSFILE ) TOFILE(BIBDESC/CLIENTP1 )
FASSFILE UF A E          DISK
F          Rename(CLIENTF1 :ASSFILEFMT)
```

```

F                                     Prefix(FI2_)
IASSFILEFMT
I           KCLIASS                    FI2_$K0011
C           Movel      '*NO '          CreateAssF          4
C*
C*****
C* Field CLIASS      : Amount difference of Euro in Franc insurance
C*           Format T          Type P(10,0)
C*
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C* Calcul. DIFFERENCE field and file field allocations before file UPDATE.
C           CALL      'CVZECRFI'
C   CLIAAS      PARM      CLIASS      WCVTIN          30 9
C   CLIASS      PARM      CLIAAS      WCVTAL          30 9
C   CLIAES      PARM      WCVTEC      30 9
C           PARM      0              WCVTDC          2 0
C           Z-ADD      WCVTEC      FI2_CLIASS
C           Eval      CreateAssF = '*YES'
C*****
C* Zone CLICAF      : Amt difference euro in f T.O.
C*           Format T          Type P(15,2)
C*
C* Lines automatically inserted by ARCAD-FRM - copyright ARCAD Software
C* Calcul. DIFFERENCE field and file field allocations before file UPDATE.
C           CALL      'CVZECRFI'
C   CLIAAF      PARM      CLICAF      WCVTIN          30 9
C   CLICAF      PARM      CLIAAF      WCVTAL          30 9
C   CLIAEF      PARM      WCVTEC      30 9
C           PARM      2              WCVTDC          2 0
C           Z-ADD      WCVTEC      FI2_CLICAF
C           Eval      CreateAssF = '*YES'
C*****
C           If      CreateAssF = '*YES'
C           Eval      FI2_CLICOD = CLICOD
C           Eval      FI2_$K0011 = CLIASS
C           Write      ASSFILEFMT
C           Endif
C           UPDATE      CLIENTF1
  
```

The called program (CVZECRFI) is given the task of carrying out the conversion.





# INTERNAL DESCRIPTIONS

# 29 Internal descriptions

## Chapter Summary

|  |     |
|--|-----|
| 29.1 Working with these cases in ARCAD Transformer Field.....            | 146 |
| 29.2 Using processes in internal descriptions.....                       | 147 |
| 29.3 Working with multi-format files.....                                | 149 |
| 29.4 Understanding file naming conventions in internal descriptions..... | 149 |
| 29.5 Locating applications in native IBM i-mode.....                     | 150 |
| 29.6 Locating applications in mode-36.....                               | 150 |
| 29.7 Referring to a file field by positional record.....                 | 152 |
| 29.8 Re-describing D.B. files.....                                       | 152 |

**Internal Descriptions** – The description of file buffer fields created directly in the programs (or in COPY clauses). These could be data files (physical, logical) or display files (DSPF) or print files (PRTF or output without any PRTF).

**External Descriptions** – The description of file fields created from the file object at the time of program compilation.

The 5 cases shown in the below table can arise:

| File          | File in the program           | TRANSFORMER type file |
|---------------|-------------------------------|-----------------------|
| D.B. File     | External description          | PF, PF38, LF, LF38    |
| D.B. File     | Internal re-description       | PF, PF38, LF, LF38    |
| Non-D.B. File | Internal description          | PF36                  |
| Non-D.B. File | Internal description, mode 36 | PF36                  |
| No file       | Internal description (COBOL)  | PF-INT                |

*Table 36: Cases to work with ARCAD Transformer Field*

## 29.1 Working with these cases in ARCAD Transformer Field

- The case of a complete database programming.  
 FLDDCT gives you the list of LSTDBFFLD fields.  
 A propagation **not** requiring an examination of the internal description (\*).
- The case of actual database files, but with programs directly re-describing them.  
 FLDDCT gives you the list of LSTDBFFLD fields.  
 A propagation **requiring** an examination of the internal description (\*).
- The case of non-database files (containing just one field (BUFFER) or two fields KEY and OTHER).

`FLDDCTS36` gives you the list of LSTDBFS36 fields.

A propagation **requiring** an examination of the internal description (\*).

4. The case of non-database files in mode 36.

`FLDDCTS36` gives you the list of LSTDBFS36 fields (with a file link in OCL36).

The propagation examines the internal descriptions by default (\*).

5. Files absent from the programs (buffer transmitted a file read/write program).

`AFLDDCTINT` and `AFLDDCTIN2` simulate the **buffer** descriptions compared to the external files.

A propagation not requiring an examination of the internal description (\*).



#### Reference

For more information about the non-database files, refer to [Working with non-database files on page 154](#)

### (\* Display/printer files

Whichever way the files are described, the display and printer files can be described as such:

- a. External description in the programs (of DSPF and PRTF).  
Purely external description programming.
- b. DSPF, DSPF36, PRTF exist but the buffer re-description is directly in the programs.  
Examination required for internal descriptions.
- c. Printer Output described directly in the programs without any corresponding PRTF.  
(Printout Card O in RPG...)

In the Transformer suite, this is compared to a fictive type PRTF36.

To sum up, if the data files and/or the display or printouts can be internally described, you must indicate **Examine internal descriptions = \*YES** at the propagation.

## 29.2 Using processes in internal descriptions

When there are internal file descriptions, the processes are carried out in the following ways at the propagation and automatic modifications:

- a. **ACVTPGMFLD – Step 1 (Propagation: Entry in the programs)**

For each program where the files are used in internal descriptions (in RPG, RPG36, RPGLE, CBL, CBLLE, CLP, OCL36), a correspondence is established between:

- The description of file fields (D.B. = PF, LF or non-D.B. = PF36)
- The description of file fields in the Program (Card I and Card O in RPG..., Buffer descriptions in CBL, Sort Cards in OCL36, Sort Cards in the FMTDTA, utilization of record position in INCCCHAR with CPYF in CLP...)

With this, the impacted program fields are only found from the position of the field in the record.

The internal descriptions are either in the main source, or in an included source.

In the case of PF, the positions in the LF can be different if the LF only takes a few file fields or if it takes fields from several physical files.

In the case of PF36, the files and index have the same description (except the keys).

**Note**

The new modified files must not have already been compiled. The files online (in the case of PF, LF) must be the old files.

**b. ACVTPGMFLD – Step 2 (Propagation and Flagging)**

Just as the normal processes, this step also flags (without modification):

- The definitions of fields that are the internal descriptions of files in the sources.  
(Card I and O in RPG, buffer descriptions in COBOL, ...)
- Detects the fields not supposed to be in the data files (and places them in the anomaly list of the non-initial file fields; in this list, they are each given a file type, “PF36” if the file is unknown, or PF or LF if it was found in the repository).

**c. ACVTPGMFLD – Step 3 (Automatic modifications in the programs)**

In addition to the standard processes, this step carries out the following modifications:

For the initial file fields:

- In RPG..., position modifications at the beginning and end of extended fields (Card I or O).
- In CBL..., length modifications of extended file fields.
- In RPG..., possible addition of new fields (with beginning and end positions) (Card I or O).
- In CBL..., addition of new fields in the file buffers.
- In RPG..., shifting of all beginning and end positions in the I or O cards if there was a field addition or extension in the lower positions.
- In RPG..., record length modification, key position and length in Card F.
- In CLP..., record positions modifications (CPYF order with INCCCHAR...), position modifications in the FMTDTA sort cards (however, no line addition in the case of field addition).
- In OCL36..., position and length modifications in the BLDFILE, BLDINDEX orders, and the OCL36 sort cards (in the case of field addition, the positions are shifted but no line is created for the new fields).

 **Important!**

Of course, if no fields to propagate are already present in cards I and O, the positions of the other fields are shifted anyway if a modified field exists in the file.

## 29.3 Working with multi-format files

When working with a multi-format file, there are no modifications for the internal descriptions at this level:

- Display/printer file fields.
- Non-anticipated fields (present in the anomaly list).

a. **ACVTDDSFLD – Modification of DSPF/PRTF**

The display fields and printouts are modified in the usual manner.

However, the internal descriptions of these are also modified in order to correspond with the modifications made in the display fields and printouts (position shifts, field extensions, field additions).

The routines inserted before display or after input, use the names of fields present in the new internal descriptions of these files.

b. **ACVTDDSFLD – Modification of interface files**

If requested, this option only carries out a process of inserting routines before write or after read (by using the names of fields present in the program).

c. **ACVTDBFFLD – Modification of PF sources**

This engine does not concern the internal descriptions. However, if your files exist as PF you should run it anyway to modify the D.D.S, otherwise it is of no use.

## 29.4 Understanding file naming conventions in internal descriptions

One of the most important characteristics of internal descriptions is that the file name in the program (visible from the outside) is not the same in all the programs that use one same file.

What really matters is the actual file name, even when in the programs, it is not the same.

 **Example**

The PCLIENT file on the disk is called:

- CLIENTS in PG1
- CLIENT in PG2
- PCLIENT in PG3

With ARCAD Transformer Field, you will only use the file PCLIENT and a link must be found to the internal file names in each program.

## 29.5 Locating applications in native IBM i-mode

---

### D.B or non-D.B files

There are file substitution orders in CLP which call the programs:

`OVRDBF FILE` (file name in pgm) `TOFILE` (actual file name on disk)

These OVRDBF are often situated in the CLP that directly calls the program.

There are sometimes several call levels between the two:

- CLP1 carries out the OVRDBF
- CLP1 calls CLP2
- CLP2 calls PG1
- PG1 calls PG2
- PG2 uses the file from CLP1

To successfully find the actual name of the file on disk, it is recommended to place a pre-compilation command in the program:

`H* %EXEC OVRDBF FILE` (file name in pgm) `TOFILE` (actual file name on disk).

#### Note

- You can automate the retrieval of these orders in the programs with the ARCAD command `ASCNOVRATR`. (It will find the ORVDBF in the pgs calling at one or several levels; it is absolutely necessary to then redo the cross-references of these programs.)
- When the file is described internally, this `%EXEC` command has no real use for the program compilation.
- However, it is used in the ARCAD cross-reference for memorizing the **Internal file name to pg external file name** link.
- The retrieval of these OVRDBF is unnecessary if the file is a D.B. file with detailed DDS in the fields and if the OVRDBF is situated in the CLP that directly calls the utilization program (one single call level).
- The retrieval of these OVRDBF is unnecessary if it is the name of the file on disk which is used in the program.

## 29.6 Locating applications in mode-36

---

### Non-D.B. files in mode 36

#### Example

The S01.CLI or S02.CLI file on the disk is called:



- CLIENTS in PG1
- CLIENT in PG2
- PCLIENT in PG3

**Note**

In mode 36, the names of the files on disk often have a prefix (or suffix) that contains a company code for example. However, it is the same file ???CLI.

With ARCAD Transformer Field, you will only begin with the file named \$.CLI and a link to the internal names of this file in each program must be found. (The \$ replaces the variable characters of the file name.)

There are file substitution orders in the OCL36 that call the programs:

- // LOAD PG1
- // FILE NAME-CLIENTS, LABEL-?4?.CLI, DISP-SHR
- // RUN

These substitutions are often situated in the OCL36 that directly calls the program.

Sometimes, there may be several levels between the two:

- OCL1 carries out the FILE NAME
- OCL1 calls OCL2
- OCL2 calls PG2
- PG2 uses the file from OCL1.

To successfully find the actual name of the file on disk, it is recommended to place a pre-compilation command in the program:

```
H* %EXEC OVRDBF FILE (file name in pgm) TOFILE (actual file name on disk).
```

Example:

```
H* %EXEC OVRDBF FILE(CLIENTS) TOFILE($.CLI).
```

**Note**

It is not possible to automate the retrieval of these mode 36 substitutions with the ARCAD command ASCNOVRATR.

The %EXEC command has no real effect on the program compilation.

However, it is used in the ARCAD cross-reference for memorizing the Pg Internal File name to External File name link.

The retrieval of these substitutions is unnecessary in mode 36 if the substitution is placed in the OCL36 that directly calls the program (in the most frequent cases).

The retrieval of these OVRDBF is unnecessary if it is the name of the file on disk which is used in the program.

## 29.7 Referring to a file field by positional record

This concept is used when you refer to a file while re-describing its positional record and while re-dividing the fields. These are internal descriptions of files.

The positional reference is made up of a 10-character name, written like this 'XppppLgggg':

- 1 character shows the type of buffer:
  - P - for a physical or logical file field
  - I - for a field declared as Input
  - O - for a field declared as Output
- 4 numbers (0000 to 9999) show the starting position of the field in the record.
- 1 character 'L' = Length.
- 4 numbers (0000 to 9999) show the number of characters used in the record to store the field.
  - For an alphanumeric field: it is the field length.
  - For a packed numeric field: it is the number of bytes taken to store the field.

| Field name | Field type | Description  |
|------------|------------|--|
| P0025L0012 | A(12)      | Alpha file Field of 12 Char. Position 25 to 36                         |
| P0036L0008 | P(15,2)    | Packed numeric file field of 15 digits but 8 chars. Position 36 to 43. |
| I0075L0004 | A(4)       | Input buffer file field  |
| O0045L0004 | A(4)       | Output buffer file field   |


Table 37: Examples of positional reference

## 29.8 Re-describing D.B. files

This concerns the following case of files existing as PF(38) or LF(38) with the fields described separately.

However, in certain programs, the files are re-described internally. In this case, only the following operations need to be carried out also:

1. Process, if necessary, the names of files in the programs.
2. Indicate **Examine the internal descriptions = \*YES** when running the propagation.

 **Important!**  
All the DB files are, therefore, single-format.





# NON-DATABASE FILES

## 30 Working with non-database files

---

### Chapter Summary

|  |     |
|--|-----|
| 30.1 Understanding the particularity of multi-format files.....        | 154 |
| 30.2 Building lists of non-database files – PF36 field repository..... | 157 |
| 30.3 Propagating and modifying non-database files.....                 | 160 |
| 30.4 Generating recovery programs – non-database files.....            | 160 |

When you work with non-database files it is important to know that they concern the following cases:

Case 1 – Files existing as PF(38) or LF(38) but their structure is not detailed (contains just one or two large fields).

Case 2 – The application is in mode S36: the files have no DDS.

To work with this, you must first build the repository of PF36 fields.

**Note**

The field lists in this case have PF36 as an attribute. This is a fictive attribute (this type doesn't exist on the AS/400). It can concern the files not used in mode S36 (Case 1).

It is necessary to indicate **Examine internal descriptions = \*YES** when running the propagation (in case 1).

**Note**

In mode 36 only the following are managed (Case 2): RPG36, RPT36 and OCL36.

CBL36 is not managed in ARCAD Transformer Field.

### 30.1 Understanding the particularity of multi-format files

---

Certain files concerned are also multi-format:

- The records are divided differently according to the value present in a position.

**Example**

Command header if 0 is at position 20.

Command details if 1 is at position 20.


This case is managed by ARCAD Transformer Field but only if you have the possibility to know which format is used.

In RPG..., the format conditions (in card I) allow you to select an adequate division.

But, in the O cards and sometimes in Card I, these format conditions are absent as they are not required for the correct functioning of the program.

To ensure ARCAD Transformer Field operates correctly, it is necessary to add *virtual format conditions* in these programs, between the file name and the first field.

- Keyword (in comment) \*%FMT in column 7 to 11 (In Card I or Card O).
- Format condition in the usual columns for this in RPG (or RPGLE) (Columns of Card I)

 **Example**  
I Cards:


```

0025.00 0031 IF3601P1 NS 01 5 C2
0026.00 I 1 2 F1AA
0027.00 I 8 13 F1ZON1
0028.00 I 14 20 F1ZON2
0029.00 I 21 30 F1ZON3
0030.00 I NS 02 48NC
0031.00 I AND 5 C1
0032.00 I OR 48NCX
0033.00 I AND 5 C1
0034.00 I 1 2 F2AA
0035.00 I 8 13 F2ZON1
0036.00 I 14 20 F2ZON2
0037.00 I 21 30 F2ZON3
0038.00 I 31 35 F2DTE1
0039.00 I 36 41 F2DTE2
0040.00 0031 IF3601P1 NS 03 48 CV
0041.00 I OR 137 CL 48NCX 5 C1
0042.00 I 8 13
F3ZON1
0043.00 I 14 20
F3ZON2
0044.00 I 21 26
F3DTE1
0045.00 I/EJECT
  
```

```

0046.00      I                27 30
F3ZON3
0047.00      I                5  5
F3I005
0048.00      I                48 48
F3I048
0049.00      I               137 137
F3I137
0050.00      I*
0051.00      I          NS  04
0052.00      I                8 13
F4ZON1
0053.00      I               14 20
F4ZON2

```

 **Example**  
O Cards: addition of format condition lines recognized in ARCAD.  
(with the I Card columns)

```

0155.00      OF3601P1 E
0156.00      O          OR                18
0156.01 ==> *%FMT   NS  01  5 C2
0157.00      O                WAAA        2
0158.00      O                W1ZON1     13
0159.00      O                20 '
'
0159.01      OF3601P1 E                20
0159.02 ==> *%FMT   NS  03  48 CV
0159.03 ==> *%FMT   OR          137 CL  48NCX  5 C1
0159.04      O                O3ZON1     13
0159.05      O                O3ZON2     20
0159.06      O                O3DTE1     26
0160.00      O          E                20
0160.02 ==> *%FMT   NS  01  5 C2
0161.00      O                WAAB        2

```

```

0161.01      O      E      20
0161.02 ==> *%FMT  NS  02  48NC

0161.03 ==> *%FMT  AND      5  C1

0161.04 ==> *%FMT  OR      48NCX

0161.05 ==> *%FMT  AND      5  C1

0161.06      O                      O2AA      2

0161.07      O                      O2ZON1    13

0161.08      O                      O2DTE1    35

0161.09      O                      O2DTE2    41

0162.00      O*

0163.00      OF3602P1 E          EXCPT2

0164.00      O*
  
```

## 30.2 Building lists of non-database files – PF36 field repository

You can build a list of non-DB fields by analyzing the internal descriptions contained in the RPG programs because there are no file descriptions. As there are no file descriptions, it is by analyzing the internal descriptions contained in the RPG... programs that you can build a list of non-D.B. fields.

The `AFLDDCTS36` command allows you to run this analysis.

It gives an output list, `LSTDBFS36` which contains all the fields of all the files found in the programs.

This command will attempt to extract the file descriptions from the RPG, RPGLE, RPG36 programs (or the /COPY). You can

- You can start the analysis, then study the output list to verify the file names and the different formats for multi-format files.

### Important!

This list must be reliable. Don't hesitate to modify the programs for the names of files in the programs (keyword `%EXEC`) or for the correct identification of formats (for multi-format files) (keyword `%FMT`).

- Redo the cross-references of modified programs.
- Then rerun the `AFLDDCTS36` to see the improved list.

### Contents of the `LSTDBFS36` list in output from `AFLDDCTS36`

- `LST_JOB` – Field name: Fictive name PxxxxLzzzz (where xxxx is the starting position and zzzz is the occupied length)

Positional reference to a file field.

- `LST_JSRCF` – Original file: actual name of file on disk (as the name in the RPG... is not reliable) (or external name defined in a %EXEC OVRDBF internal name/external name).

**Particularity:** In mode 36, if a file name is partially expressed by a replacement variable in the OCL36, it is replaced by the character \$; for the display name ?WS ?, the replacement is \$\_.



#### Example

?02 ?.CLI gives \$.CLI

?5?.CLI gives \$.CLI (same file)

TEXT?WS? gives TEXT\$\_

- `LST_TDATE` – Numeric value 9999 then 9998, etc.... (one value per file).

In `AEDTLST` and `ADSPLST`, this allows you to sort by date to obtain a sort by file(F14).

(From now on, you can also carry out sorts with the filters in `AEDTLST` or `ADSPLST`.)

- `LST_DAT` – Always 01/01/01
- `LST_CATR` – Component attribute: PF36
- `LST_JLIB` – Format: F01 by default

If the file is described with the format identifications coming from the positions in the RPG programs..., the format will be called:

- Fnnppppp where ppppp corresponds to the tested position, then C, then the compared character.
- FnnFpppp where pppp is a sequential n° (if there is more than one comparison).

In both cases you find the format conditions at the beginning of the text field.



#### Example

Format 1: if position 12=1 and position 14=2 F01F0001 Text: 12C1&14C2

Format 2: if position 12=1 and position 14<>2 F02F0002 Text: 12C1&14NC2

Format 3: if position 12<>1 F0312NC1 Text: 12NC1.

- `LST_CTYPE` – Type and length of the field A(30), S(6,0), P(11,2)
- `LST_CTXT` – Text contains the format conditions first (if there are any) followed by the character |.

Then one or several field names found in the RPG... programs.

- `LST_JXLIB` – RPG name of file (for info): One of the names given to this file in one of the RPG programs.

**Example**
**LSTDBFS36 list**

| Original File | File Attribute | Field format       | Type     |
|---------------|----------------|--------------------|----------|
| Library       | Text           |                    |          |
| ARTTARP1      | PF36 F01       | P0001L0010 A(10)   |          |
| ARTICLES      | ARCODE,        |                    |          |
| ARTTARP1      | PF36 F01       | P0011L0006 S(6,0)  | ARTICLES |
| ARTTARP1      | PF36 F01       | P0017L0030 A(30)   | ARTICLES |
| ARTTARP1      | PF36 F01       | P0047L0005 P(9,0)  | ARTICLES |
| ARTTARP1      | PF36 F01       | P0052L0005 P(9,3)  |          |
| ARTICLES      | ARPU1,         |                    |          |
| ARTTARP1      | PF36 F01       | P0057L0005 P(9,0)  | ARTICLES |
| ARTTARP1      | PF36 F01       | P0062L0005 P(9,3)  |          |
| ARTICLES      | ARPU2,         |                    |          |
| ARTTARP1      | PF36 F01       | P0067L0005 P(9,0)  |          |
| ARTICLES      | ARQTE3,        |                    |          |
| ARTTARP1      | PF36 F01       | P0072L0005 P(9,3)  | ARTICLES |
| ARTTARP1      | PF36 F01       | P0077L0005 P(9,3)  |          |
| ARTICLES      | ARPU9,         |                    |          |
| ARTTARP1      | PF36 F01       | P0082L0002 S(2,0)  |          |
| ARTICLES      | ARJJCR,        |                    |          |
| ARTTARP1      | PF36 F01       | P0084L0002 S(2,0)  |          |
| ARTICLES      | ARMMCR,        |                    |          |
| ARTTARP1      | PF36 F01       | P0086L0002 S(2,0)  |          |
| ARTICLES      | ARAACR,        |                    |          |
| ARTTARP1      | PF36 F01       | P0088L0006 S(6,0)  |          |
| ARTICLES      | ARDAMD,        |                    |          |
| ARTTARP1      | PF36 F01       | P0093L0001 A(1)    |          |
| ARTICLES      | *****,         |                    |          |
| CLIENTP1      | PF36 F02       | P0001L0008 A(8)    |          |
| CLIENTS       | CLCODE,        |                    |          |
| CLIENTP1      | PF36 F02       | P0009L0006 S(6,0)  |          |
| CLIENTS       | CLDACR,        |                    |          |
| CLIENTP1      | PF36 F02       | P0015L0006 S(6,0)  |          |
| CLIENTS       | CLDAMD,        |                    |          |
| CLIENTP1      | PF36 F02       | P0021L0008 P(15,2) |          |
| CLIENTS       | CLMCAF,        |                    |          |
| CLIENTP1      | PF36 F02       | P0029L0006 P(11,0) |          |
| CLIENTS       | CLMASS,        |                    |          |
| CLIENTP1      | PF36 F02       | P0035L0030 A(30)   | CLIENTS  |
| CLIENTP1      | PF36 F02       | P0065L0030 A(30)   | CLIENTS  |
| CLIENTP1      | PF36 F02       | P0095L0030 A(30)   | CLIENTS  |

```

|CLADR2,
CLIENTP1 PF36 F02 P0125L0005 A(5) CLIENTS
|CLCDP,
CLIENTP1 PF36 F02 P0130L0030 A(30) CLIENTS
|CLVILL,
CLIENTP1 PF36 F02 P0159L0001 A
(1) CLIENTS |*****,
```

The \*\*\*\*\* describe a fictive field positioned on the last character of the record which proves useful when generating recovery programs.

## 30.3 Propagating and modifying non-database files

### Constitution of the list of fields to process

The process to run is the same as for a list of actual database fields, except for:

- the entry list is the list of non-D.B. fields – LSTDBFS36.
- the extraction macro cannot take the field name (LST\_JOB) into account; it can, however, use RPG names of fields (present in LST\_CTXT).

#### Note

If an application is mixed, it is possible to merge this list of PF36 fields with the list of database fields for physical files to run only one propagation process.

### Propagation and modifications

#### Reference

For more information about propagation, refer to [Internal descriptions on page 146](#).

The multi-format files are managed (if those conditions that allow format recognition are present in the programs).

#### Important!

It is necessary that the descriptive file for all DB fields LSTDBFS36 is present online (check this beforehand with WRKOBJ).

To run ACVTTPGMFLD, it must be named as such.

## 30.4 Generating recovery programs – non-database files

### Particularity for non-database files (identified by PF36)

Generation of Recovery Programs

Principle: The `AGENFMTPGM` command will generate an RPGLE program for each modified file.



**Input:** The file with its old description (under the name FIINPUT with the processed fields and the rest of the records in the other fields).

**Output:** A non-indexed FIOOUTPUT file with the new description.

Each processed field will be formatted in relation to the standard routine (to go to the field in output).

In addition, a `CRTPF` process is inserted to the RPGLE program to create the file in output with its new length.

### Important!

It is necessary that the descriptive file for all the D.B. LSTDBFS36 fields be online (check this beforehand with `WRKOBJ`).

To run `AGENFMTPGM` it must be named as such.

## Generation of an associated file

An associated file can also be managed for these files without DDS.

It will have the same name as the processed file; however, DDS will be generated for this file:

### Example

```

A      R FMT
A* Primary Keys
A* Fields
A      BFL001      5P 4      TEXT ('8NC0&21CA|PUART, ')
A* Keys
A      K KEY001
A      KEY001      7A      TEXT ('8C0|NOCDE, ')
  
```

For a field to appear as a primary key in this file, you must modify the LSTDBFS36 list and enter indicate **KEY** in the column `LST_JZSEL1` selection field of the file.

Also, in the case of a multi-format file, the fields to process detected in the different formats are reused in the description of this associated file in the same record. On each creation of a record for this file, only the keys and fields to process from the current record format are populated.

## Execution of recovery programs

Run the `AEXCFMTPGM` command on IBM i. These programs create a sequential file, for each file, containing the data and the correctly-formatted data (according to the new file division).

These new files are placed in a new library, (for which you must specify the name when running `AEXCFMTPGM`). Then, you must carry out the following operations:

1. Delete the old PF36 file and its INDEX.
2. Recreate the new file with `BLDFILE` or `CRTPF`.
3. Copy the data from the formatted file to the new file with using **CPYF LVLCHK(\*NO)**.
4. Recreate the index using `BLDINDEX` or `CRTLE`. (The `BLDFILE` and `BLDINDEX` may have been automatically converted, if they were present in an OCL36.)



# OTHER FEATURES

# 31 Simulating files in COBOL

## Chapter Summary

|   |     |
|---|-----|
| 31.1 Internal description of files in the included source .....   | 163 |
| 31.2 Retrieving PF-INT fields list in COBOL – AFLDDCTINT .....    | 164 |
| 31.3 Simulating external descriptions in COBOL – AFLDDCTIN2 ..... | 165 |
| 31.4 Extracting fields to process .....                           | 165 |
| 31.5 Functioning of ACVTPGMFLD .....                              | 165 |

This is a particular case: the internal descriptions are not truly linked to a buffer file. (The access to files is carried out in other programs where the buffer is used.)

## 31.1 Internal description of files in the included source

The files do not have an external description. Their recording is described in COBOL in the included sources (COPY) in the WORKING-STORAGE section.

### Example

```

02 WS-XXXX.
03 FFS091-RECORD
.....
03 FFS092-RECORD                                PIC X
(1200) .

03 FILLER REDEFINES FFS092-RECORD.
05 FFS092-KEY.
07 FFS092-CONUMBER                                PIC 99.
07 FFS092-SEQ                                    PIC S9(3) COMP.
05 FFS092-DATA                                    PIC X(1196) .
05 FILLER REDEFINES FFS092-DATA.
07 FFS092-CONAME                                PIC X(50) .
07 FFS092-COADDR1                               PIC X(30) .
07 FFS092-COADDR2                               PIC X(30) .
07 FFS092-COADDR3                               PIC X(30) .
07 FFS092-COADDR4                               PIC X(30) .
07 FFS092-COADDR5                               PIC X(30) .
07 FFS092-COPHONE                               PIC X(20) .
07 FFS092-COCURCD                               PIC 99.
07 FFS092-COAFNO                                PIC S9(3) COMP.
07 FFS092-COCLRDAY                              PIC 9.
07 FFS092-FCVATRT                               PIC S999V99 COMP.
07 FFS092-FCVATDT                               PIC S9(7) COMP.
07 FFS092-BANKRATE                              PIC S99V999 COMP.
07 FFS092-BANKRTDT                              PIC S9(7) COMP.
07 FFS092-MMRKRATE                              PIC S99V999 COMP.
07 FFS092-MMRKRTDT                              PIC S9(7) COMP.
07 FFS092-MINSTAMT                              PIC S9(5)V99
  
```

```

COMP.
07 FFS092-MAXHIST                PIC S9(3) COMP.
07 FFS092-PRDEND    OCCURS 12 TIMES PIC S9(7) COMP.
.....

```

## 31.2 Retrieving PF-INT fields list in COBOL – AFLDDCTINT

### Retrieval of file descriptions (general principle)

Using the previous description of the file, it should extract the following description of the file:

File FFS092 Format FFS092

Fields:

```

CONUMBER                S(2)
SEQ                    S(3)
CONAME                  A(50)
COADDR1                 A(30)
.....
FCVATRT                S(5,2)
FCVATDT                S(7)
BANKRATE                S(5,3)
BANKRTDT               S(7)
MMRKRATE                S(5,3)

```

etc...

The steps for extracting this information:

1. For all the programs of the application, you should previously have built the cross-references for “the used fields” only using AUPDXREF.
2. Write a fictive source program (or several) named “PFDESC” in COBOL including all the application file descriptions in the WORKING-STORAGE section (notably by intermediate of COPY). This program doesn’t have any process but just needs to be compiled correctly.
3. Populate the cross-references for this pg “PFDESC” by UPDXREF, but this time, ask for all the fields (used or not).
4. Run the ARCAD command “AFLDDCTINT” for building the list of database fields named (LSTDBFFLD). This list is used as a base in ARCAD Transformer Field for the extraction of fields to be processed.

### Retrieval of file descriptions (details)

#### Analysis base

It focuses on the decomposition of fields for the fictive program “PFDESC”.

#### Reconnaissance of different files and file formats

Name of field of the type “filnam-RECORD”.

“filnam” will be given to the name of the File and the Format.

This means that the files are considered as having one single format.

#### Reconnaissance of different format fields

Sub-field (at the lowest level) of a “filnam-RECORD” field, on condition that it is named “filnam-fldname”.

The file field will be called “fldname” (if “fldname” contains a maximum of 10 characters).

If it contains more than 10 characters, the field file will be named using the first 7 characters of “fldname” followed by 001, 002, etc...

The character ‘-’ will be replaced with the character ‘\_’.

Its text will use the name of the field in COBOL.

#### **Example**

FFS092-MMRKRTDT -> MMRKRTDT of the file FFS092.

FFS091-ACDNRC-NF-CODES -> ACDNRC\_001 of the file FFS091.

## 31.3 Simulating external descriptions in COBOL – AFLDDCTIN2

---

The ARCAD command `AFLDDCTIN2` allows you to validate the fields detected by `AFLDDCTINT` by allocating external descriptions to all the database fields in all the programs that use them (visible by using option 14 on the program fields in `ADSPFLDREF`).

This is a simulation of external references of these fields towards fictive files. As an entry list, it uses the list of PF-INT fields extracted by the `AFLDDCTINT` command.

#### **Important!**

This process modifies the component cross-references (addition of external descriptions of fields). It is, therefore, necessary to restart it if you update the component cross-references with `AUPDXREF`.

## 31.4 Extracting fields to process

---

The extraction criteria defined in the extraction macro should be verified and possibly adapted to obtain satisfactory results.

## 31.5 Functioning of ACVTPGMFLD

---

This command will function normally, except when the files (of the PF-INT type) do not need to actually exist.

## 32 Troubleshooting

---

### Chapter Summary

|   |     |
|---|-----|
| 32.1 Modifying sources before propagation.....                              | 166 |
| 32.2 Considering the level of cross-references used for each component..... | 167 |
| 32.3 Propagating multiple applications.....                                 | 167 |
| 32.4 Avoiding incomplete operations in RPT, RPT38 or RPT36.....             | 167 |
| 32.5 Calling programs in SBMJOB by RTGDTA or by QCMDEXC.....                | 168 |
| 32.6 Transferring parameters by DS for multi-use.....                       | 169 |

### 32.1 Modifying sources before propagation

---

In certain cases, it is necessary to modify the sources before propagation and automatic modifications.

Sometimes it can be useful to manually modify the sources **before** propagation so that the propagation can allow you to validate the modifications (by detecting the inconsistencies on the nature of fields and by obtaining high quality automatic modifications), instead of carrying out these modifications **after** the propagation.

There are 3 distinct methods to carry this out:

1. Modification by a version that will be transferred to production.

The transfer to production restarts the `AUPDXREF`.

2. Modification directly in the repository.

Only if ARCAD Skipper is not used on your machine.

You should restart the `AUPDXREF` for each modified component.

3. Modification in the version where you use ARCAD Transformer Field.

You can check out the components to be modified, modify them and then compile them. However, ensure that you run an `AUPDXREF` with `VERSION(*CURRENT)` having been placed in the version. The source and the cross-references taken into account will be those of the version.

 **Warning!**

When you apply the changes (`AAPYFRMCHG`), the manually-modified source will be replaced by the automatically-modified source, based on the manually-modified source. You therefore lose the intermediate source.

## 32.2 Considering the level of cross-references used for each component

---

For all ARCAD Transformer Field processes, the cross-references of each component are taken into account. But it is possible to have several levels of cross-references for one component.

The cross-reference (and source) level present in the version is taken into account if the component was checked out and modified, with the cross-references updated and \*CURRENT in the version. The \*LASTPRD cross-reference level, the level corresponding to the last version transferred to production for each component (or the initial version if the component was never checked out) is also taken into account.

Certain cross-references can be present without affecting ARCAD Transformer Field, such as cross-references made on other open versions (not transferred to production) and that have a higher or lower version number than the version for ARCAD Transformer Field or old versions of component cross-references conserved despite the transfer to production of a new, more recent version for the component.

## 32.3 Propagating multiple applications

---

It is possible to propagate several applications at the same time, starting from the files of one or several applications. First, open a version in each application then specify the version numbers when running the ACVTPGMFLD command.

### Important!

All the other ARCAD Transformer Field commands are single-application. To work on multiple applications with other commands, go in to each corresponding version and run the other engines (ACVTDBFFLD and ACVTDDSFLLD) or the application modification commands (AAPYFRMCHG).

## 32.4 Avoiding incomplete operations in RPT, RPT38 or RPT36

---

Issues that arise when using RPT, RPT38 or RPT36 language often lead to an incorrect or incomplete operations in ARCAD Transformer Field. This language uses a pre-compiler that gives you the possibility to re-arrange the included specifications:

### Example

**PG1 in RPT** - Contains /COPY WSRCALC found anywhere in the program.

**WSRCALC (type RPT or RPG)** - Contains I cards and C cards.

The RPT pre-compiler includes the I cards and C cards in two different places in the program.

**Problem:** When loading cross-references in ARCAD (AUPDXREF), you cannot use the RPT pre-compiler because it doesn't offer the possibility of giving the cross-references of program field-lines. It is, therefore, the RPG compiler that is used. It includes all the COPY clause cards in one place which leads to compilation errors and, by consequence, incomplete or erroneous cross-references (field lengths changed to 4A, for example).

**Consequence:** In ARCAD Transformer Field, the propagation and automatic modifications are incomplete or incorrect.

**Solution:** Before loading the repository of your application, use the ARCAD command `ASCNRPLRPT` to analyze the RPT. It will indicate if it is a COPY of multi-card specifications, or even single-cards but incorrectly placed in the source.

This command also allows you to automate a process that removes this inconvenience in the RPT:

- Split the included COPY into several distinct sources (same name with card I or C, etc... at the beginning or end of the name).
- Modify the main sources by implementing COPY clauses where they belong in the new sources.

**Note**

You can leave the RPT-type on the main source as long as the compilation doesn't rearrange anything. The COPY/ clause must be situated in the place corresponding to its contents.

## 32.5 Calling programs in SBMJOB by RTGDTA or by QCMDEXC

**Problem:** Programs called in SBMJOB by RTGDTA or QCMDEXC often leads to an incorrect or incomplete operations in ARCAD Transformer Field.

The repository loading detects these program calls where the CALL PGxx is placed in a character string. You can see them by consulting the program-to-program cross-references.

Loading the cross-references also memorizes the parameters used in the calling program. In this specific case, it is a character string that is used to execute the call.

**Example**

```
CHGVAR  VAR(&CMD)  VALUE('CALL PGXX PARM(' *CAT
&VAR1 *BCAT &VAR2 *CAT `)'.
CALL    QCMDEXC PARM(&CMD 100)
```

Here, the field cross-references cannot find the names of the variables which were used for the call.

**Solution:** If you want complete cross-references (ensuring a propagation from program-to-program via the parameters), add a bit of code in this program. The code will never be executed but it will simulate a standard call.

```
GOTO AFTERCALL
CALLPGXX PARM(&VAR1 &VAR2)
AFTERCALL: . . .
```

**Note**

You can find all the programs in this case by asking ARCAD for all the programs that call the QCMDEXC or by using `ACRTRXFLST . . .` or:

```
ADSPXRF REFTYPE(*PMPGM) REFOBJ(QCMDEXC) MBRTYP(*PGM)
APPID(XXX)
```



## 32.6 Transferring parameters by DS for multi-use

**Problem:** Whatever the language, it is sometimes useful to transfer a data structure in one single parameter which is broken up according to the transmitted parameters. Transferring parameters by DS for multi-use often leads to an incorrect or incomplete operations in ARCAD Transformer Field.

**Solution:** To propagate and modify correctly, verify and modify the programs so that they respect this rule: use a **different DS when you use a list of different parameters**.

### Example

In this example there is no ambiguity between the 2 D.S. The propagation will give the correct results.

DS1: CLICOD1 from 1 to 10 and NOFAC from 11 to 18.

DS2: CLICOD2 from 1 to 10 and ARTCOD from 11 to 30.

```
CALL PG1 PARM(DS1)
MOVEL CLICOD1 CLICOD2 (To retrieve the client code)
MOVEL XXX ARTCOD
CALL PG2 PARM(DS2)
CALL PG1 PARM(DS1)
```

### Example

Because the MOVEL DS1 DS2, the 2 DS are superimposed for the propagation, this example will lead to several incorrect derived fields (ARTCOD assimilated to NOFAC).

DS1: CLICOD1 from 1 to 10 and NOFAC from 11 to 18.

DS2: CLICOD2 from 1 to 10 and ARTCOD from 11 to 30.

```
CALL PG1 PARM(DS1)
MOVEL DS1 DS2 (To retrieve the client code)
MOVEL XXX ARTCOD
CALL PG2 PARM(DS2)
CALL PG1 PARM(DS1)
```

### Example

The data structure in this example has contents of a different nature according to the situation. This will lead to several incorrect derived fields (ARTCOD assimilated to NOFAC).

DS1: CLICOD from 1 to 10 and NOFAC from 11 to 18 and ARTCOD from 11 to 30.

```
CALL PG1 PARM(DS1)
MOVEL XXX ARTCOD
CALL PG2 PARM(DS1)
CALL PG1 PARM(DS1)
```