



Adlib software documentation

© 2014 Axiell ALM Netherlands BV

Author/editor: *Erik J. Lange*

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Adlib® is a product of Axiell ALM Netherlands BV.

The information in this document is subject to change without notice and should not be construed as a commitment by Axiell ALM Netherlands. The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such a license.

Though we are making every effort to ensure the accuracy of this document, products are continually being improved. As a result, later versions of the products may vary from those described here. Under no circumstances may this document be regarded as a part of any contractual obligation to supply software, or as a definitive product description.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Table of Contents

1 Introduction	1
1.1 What is Adlib Designer	1
1.2 Adlib Designer compatibility	5
2 Screen design	7
2.1 Screens	7
2.2 Conditional screens and fields	9
2.3 Interface functionality for screen design	16
2.3.1 Accessing screens	16
2.3.2 Managing screens	18
2.3.3 Managing screen objects	23
2.3.4 Editing screen object properties	29
2.3.5 Saving modifications	31
2.4 Properties of screen objects	32
2.4.1 Screens	32
2.4.1.1 Screen.....	32
2.4.1.2 List fields.....	35
2.4.1.3 Screen conditions.....	37
2.4.1.4 Access.....	38
2.4.2 Screen fields	39
2.4.2.1 Data.....	39
2.4.2.2 Advanced.....	44
2.4.2.3 Field conditions.....	46
2.4.3 Labels	47
2.4.4 Boxes	49
2.4.4.1 Box.....	49
2.4.4.2 Advanced.....	50
2.4.5 Images	50
2.4.6 Menu options	52
2.4.7 System fields	52
2.4.8 Text windows	54
2.4.9 Web browser controls	55
2.4.10 HTML fields	55
2.4.10.1 Data.....	55
2.4.10.2 Field conditions.....	56
2.5 Adlib screens, tags and parameters	58
2.5.1 Altering screens for DOS applications	58
2.5.2 Reserved tags on screens	59
2.5.3 Tags per screen type	63

2.5.4 Parameters per screen	69
2.5.5 Designing different screen types	73
3 Application setup	79
3.1 Applications	79
3.2 Data sources	80
3.3 Methods	81
3.4 Screens	82
3.5 Output jobs	83
3.6 Export formats	85
3.7 Friendly databases	87
3.8 Users and roles	88
3.9 Database aliases (FACS)	90
3.10 Help texts	90
3.11 Interface functionality for the application setup	93
3.11.1 Accessing the application setup	93
3.11.2 Managing application objects	96
3.11.3 Editing application object properties	99
3.11.4 Saving modifications	105
3.12 Properties of Adlwin application objects	106
3.12.1 Application properties	106
3.12.2 Application titles	110
3.12.3 Authentication	111
3.12.4 Advanced	113
3.12.5 Data sources	118
3.12.5.1 Data source properties.....	118
3.12.5.2 Access rights.....	120
3.12.5.3 Methods.....	121
3.12.5.3.1 <i>Method properties</i>	121
3.12.5.3.2 <i>Access Rights</i>	133
3.12.5.3.3 <i>Screen references: properties</i>	134
3.12.5.4 Screen references.....	135
3.12.5.4.1 <i>Screen name properties</i>	135
3.12.5.5 Output jobs.....	136
3.12.5.5.1 <i>Output job properties</i>	136
3.12.5.5.2 <i>Access rights</i>	139
3.12.5.6 Export formats.....	140
3.12.5.6.1 <i>Export formats</i>	140
3.12.5.6.2 <i>Access rights</i>	142
3.12.5.7 Friendly databases.....	142
3.12.5.7.1 <i>Friendly databases</i>	142
3.12.5.7.2 <i>Access rights</i>	146
3.12.6 Users	146

3.12.6.1	User properties.....	146
3.12.7	FACS	147
3.12.7.1	Database alias (FACS).....	147
3.13	Properties of Adloan application objects	148
3.13.1	Adloan general properties	149
3.13.2	Adloan data source properties	151
3.13.3	Default values	153
3.13.3.1	Default values.....	153
3.13.3.2	Screens.....	154
3.13.4	Library branches	154
3.13.4.1	Library branch.....	154
3.13.4.2	Screens.....	156
3.13.5	Users	156
3.13.5.1	User properties.....	156
3.13.6	Fine tables	157
3.13.6.1	Fine table.....	157
3.13.7	Self-service setup	158
3.13.8	Enabling the Shelf mark column	161
3.14	Properties of screen files	162
3.14.1	Screen file properties	162
3.14.2	Screen descriptions	163
3.14.3	Access rights	163
4	Database setup	165
4.1	Databases	165
4.2	Fields	165
4.3	Multilingual fields	167
4.4	Indexes	170
4.5	Linked fields	172
4.6	Internal links	174
4.7	Reverse links	177
4.8	Feedback links	180
4.9	Domains	182
4.10	Interface functionality for the database setup	186
4.10.1	Accessing the database setup	186
4.10.2	Managing databases and datasets	188
4.10.3	Managing fields and indexes	191
4.10.4	Editing database object properties	194
4.10.5	Saving modifications	199
4.11	Properties of database objects	201
4.11.1	Database properties	201
4.11.2	Adapl procedures	208
4.11.3	Advanced	211

4.11.4	Access rights	213
4.11.5	Datasets	214
4.11.5.1	Dataset properties.....	214
4.11.5.2	Access rights.....	215
4.11.6	Indexes	215
4.11.6.1	Index properties.....	215
4.11.6.2	Advanced index properties.....	225
4.11.7	Fields	226
4.11.7.1	Field properties.....	226
4.11.7.2	Linked field.....	238
4.11.7.2.1	<i>Linked field properties</i>	238
4.11.7.2.2	<i>Relation fields</i>	249
4.11.7.2.3	<i>Link screens</i>	250
4.11.7.2.4	<i>Linked field mapping</i>	254
4.11.7.3	Enumerative field.....	259
4.11.7.3.1	<i>Enumeration Values</i>	259
4.11.7.4	Auto-numbered field.....	261
4.11.7.4.1	<i>Automatic numbering</i>	261
4.11.7.5	Image field.....	263
4.11.7.5.1	<i>Image field properties</i>	263
4.11.7.6	Application field.....	273
4.11.7.6.1	<i>Application field properties</i>	273
4.11.7.7	Multi language fields.....	277
4.11.7.8	User interface texts.....	277
4.11.7.9	Default values.....	278
4.11.7.10	Z39.50 values.....	279
4.11.7.11	Access rights.....	280
4.11.8	Internal links	280
4.11.8.1	Internal link properties.....	280
4.11.9	Feedback databases (references)	284
4.11.9.1	Feedback database properties.....	284
5	Using ADAPL	286
5.1	Programming in ADAPL	286
5.1.1	Introduction	286
5.1.2	The ADAPL editor	287
5.1.3	Compiling an adapl	289
5.1.4	Using ADAPL programs	292
5.1.5	Language basics	294
5.1.6	Reserved words	295
5.1.7	Data types	297
5.1.8	Constants	299
5.1.9	Variables	300

5.1.10	Expressions	313
5.1.11	Instructions	319
5.1.12	Functions	329
5.1.13	FACS	335
5.1.14	Handling text files	339
5.1.15	Using Word templates	342
5.1.16	Output formats	342
5.1.17	F12 adapl	343
5.1.18	Running an adapl	344
5.2	Reference	346
5.2.1	ABS	346
5.2.2	AFTER\$	347
5.2.3	ASC	347
5.2.4	ATTR	348
5.2.5	ATTRIB	349
5.2.6	BEFORE\$	350
5.2.7	BINPATH	351
5.2.8	BOX	351
5.2.9	BREAK	352
5.2.10	CHDIR	352
5.2.11	CHR\$	353
5.2.12	CLEAR	355
5.2.13	CLOSE	355
5.2.14	CLOSEALL	356
5.2.15	CLOSEALLFILES	356
5.2.16	CLOSEFILE	356
5.2.17	CLR	357
5.2.18	CLS	358
5.2.19	COLUMN	358
5.2.20	COPY	359
5.2.21	COPYFILE	359
5.2.22	CURSOR	360
5.2.23	CVT\$\$	360
5.2.24	DATDIF	361
5.2.25	DATE\$	362
5.2.26	DAYOFWEEK	363
5.2.27	DEL	364
5.2.28	DELETE	364
5.2.29	DELETEFILE	365
5.2.30	DISPLAY	365
5.2.31	DO UNTIL	366
5.2.32	END	366

5.2.33	ENUMCODE\$	367
5.2.34	ENUMVAL\$	368
5.2.35	ERRORF	369
5.2.36	ERRORM	370
5.2.37	ERROR\$	370
5.2.38	FDEND	371
5.2.39	FDSTART	372
5.2.40	FIELDISREPEATED	374
5.2.41	FIELDLENGTH	374
5.2.42	FIELDTYPE	375
5.2.43	FORMATFIELD	376
5.2.44	GETCOUNTER	378
5.2.45	GETKEY	380
5.2.46	GETVAR	383
5.2.47	GOSUB	384
5.2.48	GOTO	384
5.2.49	HTMLTOISOLATIN	385
5.2.50	IF THEN ELSE	385
5.2.51	INCLUDE	386
5.2.52	INPUT	386
5.2.53	INSTR\$	388
5.2.54	INT	388
5.2.55	IS	389
5.2.56	ISIN	390
5.2.57	ISISBN	391
5.2.58	ISISMN	392
5.2.59	ISSSN	392
5.2.60	ISMODIFIED	393
5.2.61	ISOLATINTOHTML\$	393
5.2.62	ISOLATINTOUTF	394
5.2.63	ISSTOPWORD	395
5.2.64	JSTR\$	395
5.2.65	LAUNCH	396
5.2.66	LEFT\$	397
5.2.67	LEN	398
5.2.68	LET	398
5.2.69	LOCATE	399
5.2.70	LOCK	400
5.2.71	MAX	401
5.2.72	MID\$	401
5.2.73	MILESTONE	402
5.2.74	MIN	403

5.2.75 MKDIR	403
5.2.76 MOD	404
5.2.77 NAME\$	404
5.2.78 NDAYS	405
5.2.79 NULL	406
5.2.80 ONCHANGEIN	407
5.2.81 ONEOJ	408
5.2.82 ONEOP	408
5.2.83 ONSCREEN	409
5.2.84 ONSOJ	409
5.2.85 ONSOP	410
5.2.86 OPEN	411
5.2.87 OPENFILE	412
5.2.88 OPENURL	412
5.2.89 OUTPUT	413
5.2.90 PAGE	414
5.2.91 PAGEBREAK	415
5.2.92 PDEST	415
5.2.93 PRINT	416
5.2.94 PRINTIMAGE	417
5.2.95 PROGRESS	418
5.2.96 QUIT	419
5.2.97 READ	419
5.2.98 RECCOPY	423
5.2.99 RECDATE\$	424
5.2.100 REDISPLAY	425
5.2.101 REGVALIDATE	426
5.2.102 RENAME	426
5.2.103 REPCNT	428
5.2.104 REPCOPY	429
5.2.105 REPFIND	429
5.2.106 REPINS	430
5.2.107 REPLACE\$	431
5.2.108 REPMAX	432
5.2.109 REPMIN	433
5.2.110 REPSORT	434
5.2.111 REPSORTINS	435
5.2.112 REPSUM	437
5.2.113 RETURN	437
5.2.114 RIGHT\$	438
5.2.115 RINSTR\$	438
5.2.116 RMDIR	439

5.2.117	ROUND	440
5.2.118	ROUND\$	440
5.2.119	SENDMAIL	441
5.2.120	SETFONT	443
5.2.121	SETLINESPACING	444
5.2.122	SETORIENTATION	445
5.2.123	SETPAPERSIZE	446
5.2.124	SETSTATUSBARTEXT	449
5.2.125	SETVAR	450
5.2.126	SETWINDOWTITLE	451
5.2.127	SHOW	452
5.2.128	SKIP	453
5.2.129	SQRT	454
5.2.130	STATUS	454
5.2.131	STR\$	455
5.2.132	STRING\$	458
5.2.133	SWITCH	458
5.2.134	SYSTEM	461
5.2.135	TAG2FIELD\$	463
5.2.136	TEXT\$	463
5.2.137	TIME\$	464
5.2.138	TITLE	464
5.2.139	TOSTRING\$	465
5.2.140	TRANSFORM	466
5.2.141	TRIM\$	467
5.2.142	UNLOCK	468
5.2.143	USER\$	468
5.2.144	VAL	469
5.2.145	WAIT ... NOWAIT	470
5.2.146	WHATDATE	471
5.2.147	WHILE	472
5.2.148	WORDCREATEDOCUMENT	472
5.2.149	WORDCREATELABELS	477
5.2.150	WORDPRINTDOCUMENT	479
5.2.151	WRITE	480
5.2.152	WRITEEMPTY	481
5.2.153	YESNO	482
6	Managing translations	483
6.1	The Translations manager: introduction	483
6.2	Interface functionality for translations	484
6.2.1	Accessing the Translations manager	484
6.2.2	Editing and translating interface texts	486

6.2.3 Saving your work	489
7 Import and export	491
7.1 Exchanging data: introduction	491
7.2 Interface functionality for import and export	494
7.2.1 Accessing the job managers and editors	494
7.2.2 Managing import and export jobs	495
7.2.3 Editing job properties	497
7.2.4 Saving modifications	499
7.2.5 Running import and export jobs	500
7.2.6 Using batch jobs	502
7.3 Properties of import jobs	505
7.3.1 General	505
7.3.2 Mapping	506
7.3.3 Options	511
7.3.4 Advanced	524
7.4 Properties of export jobs	526
7.4.1 General	526
7.5 Exchange formats	529
7.5.1 Tagged ASCII (Adlib)	529
7.5.2 DBASE III/IV (*.dbf)	530
7.5.3 ASCII delimited (*.csv)	531
7.5.4 ASCII fixed length	532
7.5.5 PICA III	533
7.5.6 MARC (general ISO 2709)	537
7.5.7 MARC (Ocelot)	538
7.5.8 MARC (CDS-Isis)	538
7.5.9 XML (general, *.xml)	538
7.5.10 MS Excel (.xls/.xlsx)	539
7.5.11 Image directory	540
7.5.12 Modes	555
7.5.13 Other formats/separators	556
8 General topics	558
8.1 Adlib file types and folders	558
8.2 Searching for Adlib objects	564
8.3 Backups, and logging and recovery	565
8.4 User authentication and access rights	572
8.5 Status management of authority records	584
8.6 Managing record locks	586
8.7 Colour your Adlib application	588
8.8 Windows Image Acquisition	591
8.9 Using a web browser control	593

8.10	Using an HTML field	597
8.11	Showing the upwards hierarchy of a linked term	599
8.12	Character set conversion of your data and/or application	600
8.13	Inserting special characters in text	603
8.14	Working with non-standard dates	604
8.15	Naming fields for hierarchical display	605
8.16	Regular expressions	606
8.17	Domain-specific icons	609
8.18	HTML start page for data source selection	612
8.19	Approaching external sources as friendly databases	614
8.20	Requirements for the Change locations procedure	616
8.21	Testing your application for errors	618
8.22	Error codes	620

1 Introduction

§

1.1 What is Adlib Designer

The Adlib Designer toolkit has replaced ADSETUP and DBSETUP for creating and editing Adlib applications and databases.

There are a number of tools available, some of which replace only a specific aspect of application management, previously found in either one of said older tools. The new tools are listed in the *Tools* and *View* menu but may also be opened from the toolbar.

There is no general way of working with Adlib Designer. Some tasks have one specific tool, like managing record locks, while for others, like editing a database definition or an application definition, you have to open the *Application browser* and search for the objects* that you want to edit.

* Adlib objects are the interchangeable elements that, together, make up an Adlib application: most of the nodes in the *Application browser* are objects, including folders. All objects have properties or attributes. So for instance: a field is an object, and its name and length are properties of that object. (Some properties are read-only.)

Select a work folder

To ease working with the several tools, you may select your work folder in the main *Designer* window first. Select for instance the (copy of the) Adlib folder you want to work in. Also, Adlib Designer remembers your current general work folder and the work folders selected for the individual tools the next time you start the program, so you don't have to select them again. The selected work folder is displayed in the status bar of tools windows.

Interface and application language

Via the *Application language* menu, select the language in which you want to view the objects which comprise your application, in list views and in the *Screen editor* in Adlib Designer: data sources, access points, field names, labels and output formats and such, will then be

displayed in this language (if texts in that language are available of course). Regardless of the application language you choose, you'll be able to view and edit all translations of fixed texts where applicable: to set the application language to your own language just eases your work in Adlib Designer.

Underneath the *Options* menu you can set the language for the user interface of Designer (its menus and property labels). It may be convenient to set this option to your own language, but note that the Designer Help you are currently reading, may not be available in that language.

Available tools

The currently available tools are the following:

	<p>Application browser: with the <i>Application browser</i> you can scroll through your application similar to scrolling through folders and files on your computer with Windows Explorer. But in the application/object browser you'll only see folders and Adlib objects, like screens and application definitions and databases. From here, you can add new Adlib objects too, to the folders of your choice. And when possible, you'll find the properties of a selected object in the right pane of the application browser, where you may edit them.</p> <p>Introductory Help topics:</p> <ul style="list-style-type: none">Accessing the application setupAccessing the database setup
	<p>Screens manager and Screen editor: these tools allow you to manage and edit all your Adlib screen files or create new ones. (But use the <i>Application browser</i> to link screen files to an application.)</p> <p>Introductory Help topic:</p> <ul style="list-style-type: none">Accessing screens
	<p>Import- and Export job manager and editor: manage, create, edit and execute import or export jobs from their respective tools.</p> <p>Introductory Help topic:</p> <ul style="list-style-type: none">Accessing the job managers and editors

	<p>Translations manager: the <i>Translations manager</i> is for viewing, editing and/or translating all texts in your Adlib system in one overview (except for the Help files, at the moment). In the overview you can easily find any text in any file, all at once: just sort on a column of your choice and scroll to the texts you are searching for.</p> <p>Introductory Help topic: Accessing the Translations manager</p>
	<p>Recovery tool: use the <i>Recovery tool</i> to repair lost data, by importing a logging file that contains all changes to your records since your last backup.</p> <p>Full Help topic: Backups, and logging and recovery</p>
	<p>Object searcher: use the <i>Object searcher</i> tool to search your application for names of Adlib objects, that appear in any of four possible file types. With the search result as reference, you can edit the application (in the <i>Application browser</i>) without the risk over overlooking any reference.</p> <p>Full Help topic: Searching for Adlib objects</p>
	<p>Application tester: check your application for some specific errors.</p> <p>Full Help topic: Testing your application for errors</p>
	<p>Record lock manager: with this tool, you may keep real-time track of locks applied to records when they are edited. Erroneous locks can be deleted.</p> <p>Full Help topic: Record locks</p>
	<p>Application character set conversion tool: you can change the encoding of database structures (<i>.inf</i> files), application structures (<i>.pbk</i> files) and screens (<i>.fmt</i> files) simultaneously, with the <i>Application character set conversion</i> tool.</p>

	<p>Full Help topic: Character set conversion of your data and/or application</p>
	<p>Change your application colours tool: apply a colour scheme to an entire application at once, or rather, to all screens in a folder that you select.</p> <p>Full Help topic: Colour your Adlib application</p>
	<p>Batch job manager: create, manage and execute import or export job batches.</p> <p>Full Help topic: Using batch jobs</p>

The main Adlib Designer window

The output of many tools in Designer is directed towards the document space in the main Adlib Designer window. When you have just started Designer, this document is empty. What the output consist of, depends on the tool: sometimes it may be a search result, but mostly it will be a report about an executed process (including any errors that may have occurred). All output will be inserted beneath the previous output, unless you empty the result document by clicking the Delete messages button:



You can edit the output in the result document by typing or deleting text in it. You may even paste copied images or text from other Windows programs into this document. You can save the result document at any time by choosing File > Save result window as, or by clicking the Save as button:



Find in results

In the main *Adlib Designer* window choose *Edit > Find (ctrl+F)* to search for a term in the result document.

Type any term or part thereof you want to search in all of the text

displayed in the main *Designer* window.

If the term you type is only part of a word or words you look for, then deselect the *Match words* option.

Leave the *Match case* option unmarked if upper and lower case are not important while searching.

Click *OK* to start the search. A found term is highlighted. To search a next appearance of the searched term, each time press **F3**.

Important!

To ensure the safety of your current application and data, always make a copy of your application and maybe of your data too (the latter only if no-one will be working on it while you edit the application), and only edit that copy. When you are certain your edited copy is working okay, you can replace the original with it. Also make regular backups of your original Adlib folder, so you can always go back to a previous state of the application.

1.2 Adlib Designer compatibility

Adlib Designer is compatible with all versions of the Adlib executables, and runs on all Windows versions from 98 up, but Adlib recommends not to use Windows versions older than Windows 2000 or Vista (this is a requirement for the *Record lock manager tool* in Designer, because this tool needs to monitor the file system permanently, and Windows versions older than 2000 do not support this file system monitoring). This means you can edit your application in Adlib Designer just as well when you are still using Adlib version 4.5.4, for example, as when you use the most recent version. And when you save this edited application, it will be compatible with the same software version 4.5.4*. So you need not upgrade your Adlib executables to run an application that you edited in Adlib Designer.

The other way around also works, so applications edited with Adlib Designer 6.0 for example, can be run by Adlib executables higher than 6.0.

There are a few issues to take into account though:

* If you are not using the most recent version of the Adlib executables to run your applications, Adlib Designer saves your application definition en database structures (also imported files) in a format that can be run normally by your older executables. But Designer still offers settings for Adlib objects that are newer than the version of your executables. If you were to use these newer settings when editing your application, Adlib Designer automatically upgrades said structures to the version that includes the added (non-default)

setting, after which you can no longer run the application with your version of the executables until you reset the newer setting to its default value again through Designer. And Adlib Designer does not warn you when you use such a newer option. Unfortunately this means that you have to check several release-notes to be sure that an option you now want to use for the first time does not upgrade your files inadvertently. Therefore, Adlib recommends to upgrade your executables to the most recent version, to be able to apply the full range of settings that Adlib Designer offers, and to enjoy all the new functionality added in the newer executables.

2 Screen design

§

2.1 Screens

A "screen" is Adlib jargon for the screen tab or tab sheet or form which displays a list of retrieved records in the so-called *Brief display* (in a running application), or the screen tab that shows part of the data from a single record in the so-called *Detailed display* of a record (usually several screens are used to present all data from a record). You'll also find screens in use as the *Query by form* window (a.k.a. *search screen*) for advanced searches, as *Zoom/edit* windows to display a small selection of data from a linked record in another database, and as a so-called *Link screen* to determine part of the layout of a list of retrieved terms for a validated field on the *View table* tab in the *Find data for the field...* window. (See the Adlib User guide for information about where these displays show up in a running application.) Each type of screen has its requirements and therefore has its own particular design characteristics. [Click here](#) to learn how to design screens, and link them to your application.

Screen names

Screens are separate files and have the extension *.fmt* (format). When you name a screen, you can give it any unique name you want, but in our model applications we often use certain prefixes to be able to see quickly what the purpose of a screen is, when you are browsing the file names:

- *Brief display* screen names often start with *br_*, for instance: *br_catal.fmt*.
- *Query by form* screen names normally begin with *qbf*, like: *qbfbook.fmt*.
- *Link screen* names usually start with *lnk_*, for instance: *lnk_copy.fmt*.
- *Zoom/edit* screen names often start with *zm_*, for example: *zm_loan.fmt*.
- The rest of the name, or the first part of the name if the screen is used for detailed display, is often determined by the name of the data source from which data is displayed in this screen, like:

zm_copy.fmt or *docrepro.fmt*.

See the Help topic *Managing screens* for information about how to create new screen files and where to find existing screens.

Applying screens

To specify which data can be entered or must be displayed on a screen in a running application, the designer of the screen places different (visual) screen elements called screen objects on the screen, in the *Screen editor*: it concerns objects like entry fields, labels, boxes (frames), menu options (checkboxes), and image place holders.

Most of such objects must then be associated with database fields or adapls for instance, through their properties.

And the screen file itself must of course be linked to the application definition. Per data source or per method in that application definition, you list all screens (actually, references to the screen files) that must be used for the detailed presentation of records, for the brief display, and as search screen. To indicate the difference between screen references and the actual screen files, different icons are being used:

- Screen files have the following icon:



An application folder may contain screen files that are meant for use in that application only, but the presence of those files doesn't mean that they are actually linked to the application: they are just Adlib screen files in a folder, and in newer applications you will only find those screen files in the `\screens` folder.

- References to screens have any of the following icons:



References to screen files are part of an application definition. You specify these references for data sources and/or methods in the tree view in the *Application browser*, to determine exactly which brief screens and which detail screens should be used to either display the search result list, to display data from one record in the data source, or to display a particular *Query by form* for searching.

Before editing a screen...

In principle, you can edit all existing screens in your application, or add new ones. But before you do so, please note the following:

- Back up your existing screens, or better, your entire Adlib folder, so that if anything goes wrong, you can always go back to a working state of your application. Click [here](#) for more information about safe backup and logging procedures.
When you've made a backup of your Adlib folder, and you mess up editing an existing screen, you can simply copy the old screen from your backup to the Adlib folder you're working in, using the Windows Explorer or the *Application browser* in Designer.
- A screen file may be in use for different data sources, so its design must apply to all of those.
- You can use any existing screen as the basis for a new screen. Just open a screen which looks a lot like the new screen you want to make, edit it to your liking, and save it under a new name.
If you change the name of an existing screen, the old screen will of course continue to exist and Adlib Designer will simply create a new screen file with the new name.
- You need to know the purpose and meaning of a few so-called *reserved tags* when you want to edit brief display screens.
- If you're still working with Adlib applications for DOS, know that there are certain screens that need special care when editing them: click [here](#) for more on this topic. Also, there are *parameters* and a lot of *reserved tags* that you may come across.

2.2 Conditional screens and fields

The purpose of conditional screens and fields is to be able to handle screens and fields more dynamically, to hide or show a screen or field automatically, for instance if the user enters a certain value somewhere. This allows you to show little used screens or fields only if the need arises, and with it, keep the user interface somewhat simpler.

Suppressing fields and screens this way is not a safe way to protect confidential information, because there would still be other ways to extract the information you want to protect. Confidential information is better protected with the proper access rights.

From 6.6.0, conditions for hiding a screen can be set in the properties of the screen itself. The setup of a conditional field can be found in the properties of the screen field which can be hidden or set to read-only. The condition which you can provide to hide or show a screen or field, must be shaped in the form of a single expression or a plural, Boolean expression. Every time you open a record for display or editing, and every time you leave a field in edit mode, all relevant

expressions will be evaluated. If a screen condition is true, the screen will be shown, while if the result is false, the screen will be hidden. To field conditions, the opposite applies: if the expression evaluates to true, the relevant field will be set to read-only or will be hidden.

In the expression you can use tags and/or English field names, comparison operators like =, >, <, >=, <=, Boolean operators, literal values, and brackets () to control the order of evaluation.

Syntax

The basic form of a single expression is:

```
<field or constant> <operator> <value>
```

For example: `material = 'book'`. The condition is case-sensitive, so `'Book'` is different from `'book'`. Single expressions can be combined into plural expressions, using Boolean operators. In an expression you can use the following elements:

- **Field tags** and/or English field names. An Adlib field name cannot have spaces in its name, and both tags as well as field names are case-sensitive. Do not use linked fields.
- **Constants** are a kind of system variables which contain values from the system or the record. Constants can be used instead of a field name if they produce a value, or used by themselves as the entire expression if they generate a Boolean value directly. The available constants are:
 - `$priref`, contains the current record number;
 - `$locked`, evaluates to true if the record is locked;
 - `$newrec`, evaluates to true if this is a new record;
 - `$role`, contains the role(s) of the current user;
 - `$admin`, evaluates to true if the user's current role is \$ADMIN.

Some examples:

```
$priref > 1000
```

Result: the screen will be displayed if the record number of the current record is greater than 1000.

```
$newrec
```

Result: the screen will be displayed if the record you just opened is a new record.

```
$role != student
```

Result: the screen will be displayed if the role of the current user is unequal to `student`. `$role` returns an empty string if for the current user no role has been specified, in which case the result

of this expression is true.

- **Comparison operators** compare two values, and produce a Boolean 'true' or 'false'. When comparing two values, they should be of the same data type. So preferably compare the value from an integer field to an integer value, and compare a text field to a string, etc. You can use any of the following comparison operators:

= (equals)
!= (not equals)
> (larger than)
< (smaller than)
>= (larger than or equal to)
<= (smaller than or equal to)
~ (compare to a regular expression)

If a field has more than one occurrence and/or contains multilingual data, then actually not just two values will be compared, but a set of values will be compared to a single value or, when you are comparing two fields, two sets of values. However, from the two sets only two single values have to satisfy the equation to make the expression true. In other words: the expression `material = 'book'` is already true if one of the occurrences of the field, in one of the data languages, contains the word `book`; the values in the other occurrences and data languages are not relevant. By the way, you cannot filter by occurrence or data language, so in the expression it's not possible to indicate an occurrence number or data language.

Advanced comparisons: regular expressions

A tilde (~) must be followed by a regular expression enclosed in quotes. A regular expression is a special string which indicates how a value can be formatted. Typically, you would use a regular expression if a field may contain many different values which nonetheless have some similarities, like for instance the object number field in which all values (or a specific part of all values) start with certain characters or contain a number sequence. All Dutch postal codes for example (without a space), satisfy the following regular expression: `[1-9][0-9][0-9][0-9][A-Z][A-Z]`. With the full expression: `address.postal_code ~ '[1-9][0-9][0-9][0-9][A-Z][A-Z]'` you could hide a field or screen if the postal code field contains a postal code which satisfies the regular expression. In the Regular expression topic you can find more information about the syntax of regular expressions.

Advanced comparisons: different data types

If the data types of values on both sides of a comparison operator mismatch, you can still compare them, but the evaluator will cast the values to a common data type. The “simpler” data type of the two values will become the common data type, in the following order: numerical -> integer -> (text) string -> Boolean. So if you compare a numerical value like 1.25 to a string like 'book', both values will be regarded strings and will be compared alphabetically. When comparing two field values however, it is mandatory that both fields have the same data type.

- **Boolean operators** can be used to combine single expressions. In fact, each single expression results on evaluation in a Boolean value ('true' or 'false'): a Boolean operator compares two Boolean values and leaves just one Boolean value. You cannot use the commonly known AND, OR and NOT operators here; instead you must use:

&& representing AND, true if both expressions are true;
|| representing OR, true if at least one of the expressions is true;
! representing NOT, true if the first expression (to the left of !)
is true while the second expression (to the right of !) is false.

Some examples:

```
$priref > 1000 && $role = financial
```

Result: the screen will be displayed if the record number of the current record is greater than 1000, and the role of the current user is financial.

```
material = 'book' || material = 'serial'
```

Result: the screen will be displayed if the material field contains either the value book or the value serial.

- **Brackets ()** control the order of evaluation, if it must be different from the normal rules for operator precedence. It works like any mathematical expression and the comparison operators are always evaluated before any Boolean operators. Typically, you only need to use these brackets if your expression contains two or more Boolean operators and the standard left-to-right evaluation of the expression is not what you want.

Example:

```
$newrec && (material = 'book' || material = 'serial')
```

Result: the screen will be displayed if the material field contains either the value book or the value serial, while this record is new.

The result of the Boolean combination of the second and third

expression will be combined with the Boolean result of the first expression. If you leave out the brackets, the Boolean result of the first and second expression will be combined with the Boolean result of the third, which may have different results.

- **Value** is an integer (a whole number without dots or commas, like 6, 100003, etc.), numerical (a broken number with a decimal point, like 1.25, 23.0009, etc.) or a string (character sequence, e.g. 'this is a string'). Preferably enclose a string by single or double quotes, although that is only mandatory if there are spaces in the string. If you put a string in between quotes, then at least it can't accidentally be interpreted as a field name. Strings can be truncated by placing an asterisk behind the string, but still in front of the last quote, like: 'strin*'. So there is no implicit truncation like in the *Search wizard* of Adlib. Dates can only be compared as strings (alphabetically), which means you can only compare ISO dates correctly, since 2010-02-01 is indeed "greater than" 2010-01-10, but the European date 01-02-2010 is "smaller than" 10-01-2010 in an alphabetic sense. Keep this in mind if you include date fields in your expression. Preferably, use no values from linked and merged-in fields in your expression. The reason is that data which is retrieved alongside linked fields will only be retrieved when it is necessary, for instance only when in detailed display of a record you switch to a tab which contains the linked field. This might mean that in this case a conditional screen can't be hidden at the right moment. Another remark: do not copy (example) expressions from Microsoft Word documents, since the standard quotes used in those documents (") are not suitable for use in the conditional expression: therefore type your expression manually in Designer and you will automatically use the correct quotes.

As soon as you change a field in a record and leave the field, all field and screen conditions (if present) will be evaluated, and fields and screens will be hidden or shown as appropriate.

Precedence of hiding

Since there are different ways of hiding a screen, it must be clear which method takes precedence. Succinctly put: access rights come before the manual hiding of screens which in turn comes before conditional hiding. This has been implemented as follows:

- If access rights prohibit a screen from being shown, then you can't switch it back on via the *Select screen tabs* button submenu, nor can it be displayed again via a condition set for the screen.
- If access rights do allow a screen to be shown, but you hid the

screen via the *Select screen tabs* button submenu, then it cannot be displayed again via a screen condition until you manually switch the screen back on again.

- If a screen is hidden by a condition set for it, you can't display it again via the *Select screen tabs* button submenu.

Error handling

Expressions which cannot be evaluated because of an error, will usually generate a message in your Adlib application, but errors won't always be recognized as such by the evaluator. This is because all elements in an expression which are not recognized by the evaluator as either a tag, an English field name, a constant or operator, will be interpreted as a string. So if you were to use the expression `name_type = 'SUPPLIER'`, then Adlib would not recognize `name_type` as a field name because the proper field name is `name.type`. That is why `name_type` will be interpreted as a string, just like 'SUPPLIER'. In this case, that will always generate the Boolean value 'false', since the two strings are permanently different, and the relevant screen will always be hidden.

An error causing an error message makes Adlib ignore the expression so that the relevant screen is displayed by default. Preferably, the error in the conditional expression should be solved immediately, so that no more error messages appear. By the way, there's a chance that an error message is shown twice, but that is currently inherent to this functionality.

Example setup

In the *Persons and institutions* data source in the Acquisitions module of the 4.2 model application, there are three detail screens visible by default: *Name information*, *Supplier details* and *Management details*. If you use the Acquisitions module as an integrated part of a Library application, there's a good chance *Persons and institutions* will contain not only suppliers, but also publishers, authors, institution names, etc. And even when you are accessing *Persons and institutions* from the Acquisitions module, you can search for names in all of the domains (name types). When displaying or editing the name data of an author for example, you may find that the presence of the *Supplier details* tab is superfluous or even confusing. It would be nice if this tab would only show if (one of the occurrences of) the *Name type* field would contain the value *supplier*.

1. In the Application browser, open the *library acquisitions* folder. Underneath the Adlib Acquisitions 4.2 pbk, you'll find the *Persons and institution (suppliers)* data source, and in it the *Screens* list.

2. Double-click the *Name information* and the *Supplier details* screen to open them in the *Screen editor*. On the *Name information* screen we can observe that the tag of the field on which we are going to base our condition is “do”.
3. In the *Fields* list in the *People* database in the data folder – *Persons and institutions* is the user-friendly interface name for the *People* database – you can find out that the *do* tag is an enumerative field named *name.type*, and that the neutral value for *supplier* (a translated value) is *SUPPLIER*. We need this neutral value as well.
4. Go back to the *Screen editor* and select the *Supplier details* screen. Since it is this screen that we wish to show conditionally, we must enter the condition in the properties of this screen.
5. Right-click an empty spot in the screen (not inside a box or field), and choose *Properties* in the pop-up menu.
6. The *Adlib object properties* window opens. Select the *Screen conditions* tab.
7. You must now manually enter the condition for showing this screen. A conditional screen will be hidden if the evaluated condition is false, and will only show if the condition becomes true. In spoken language our condition must then be: if the *name.type* field contains the value *SUPPLIER* then show this screen. So it will be hidden by default. In the condition syntax, this translates into the expression: `name.type = 'SUPPLIER'`. However, equally well would be: `do = 'SUPPLIER', name.type = SUPPLIER`, or `name.type = "SUPPLIER"` for example.
8. Close the *Adlib object properties* window and save the changes in the screen.
9. Restart your Adlib Acquisitions module, open successively the detailed display of suppliers and of persons who are not a supplier, and observe that the *Supplier details* screen will only be shown if the record pertains to a supplier.
Also when you edit a record, and remove or add a *Name type* occurrence containing *supplier*, the *Supplier details* tab will be hidden or shown accordingly when you leave the field.
10. We may also choose to have fields conditionally hidden or set to read-only. Let's take the *Supplier details* tab again as our example, which might still be opened in the *Screen editor*. Right-click the *s7* field which will hold the currency of the supplier, and select *Properties* in the pop-up menu.
11. In the Adlib object properties window, open the *Field conditions* tab and put the cursor in the *Read-only conditions* box. Type the

following expression, for example: `S6 = English && S7 = GBP`

12. Close the window and save the changes in the screen.
13. Restart the Adlib Acquisitions module and look up a supplier record. The new field condition makes sure that when you leave a field and the *Language* (S6) field contains `English` while in the *Currency* (S7) field the value `GBP` is present, the *Currency* field is set to read-only. Should you want to change the currency now, then you'll first have to enter something other than `English` in the *Language* field.
Coincidentally, both fields in this expression are present on the *Supplier details* screen and one of the fields is also the conditional field itself which is set to read-only, but that is by no means necessary of course: as long as the fields used in an expression are present in the current database.
Expressions entered in the *Suppress conditions* box can actually hide the current field (plus its label).

2.3 Interface functionality for screen design

§

2.3.1 Accessing screens

In Adlib Designer there are two ways to list existing screens and open one or more of them for editing in the *Screen editor*: through the *Application browser*, or through the *Screens manager*. In both ways it's also possible to create new screen files, as in the *Screen editor* itself.

Screens in the Application browser

With the *Application browser* you can scroll through your application similar to scrolling through folders and files on your computer with Windows Explorer. But in the application/object browser you'll only see folders and Adlib objects, like screens and application definitions and databases. From here, you can add new Adlib objects too, to the folders or lists of your choice. And for screen files and screen references, you'll find a few properties of a selected screen in the right pane of the *Application browser*, where you may edit them; the properties of screen files that are displayed here, are just a small selection of all the properties of the screen that are accessible in the

Screen editor.

Follow these steps:

1. In the main *Adlib Designer* window that opens when you start Designer, start the *Application browser* by choosing *View > Application browser* or clicking the button for this tool:



2. Select your work folder in the *Application browser* window, by clicking the *Open folder* button:



Preferably, choose your main (copy of an) Adlib folder, not one of the subfolders in it. This allows you to browse all your Adlib objects quickly. Click here for information about how to find a certain screen.

3. In the left window pane of the *Application browser*, click the **+** in front of each folder or object to expand the tree structure and display all objects or folders underneath the current item. Click **-** in front of each folder or object to fold it in. So click the desired subfolder node and just select the screen file or reference that you are looking for, to display all or some of its properties in the window pane on the right, or double-click the file to open it directly in the *Screen editor*. Note that if you double-click a screen reference, as can be found within an application definition (*.pbk* file), you of course open the referenced file. And before you edit a screen, remember that one and the same screen file may be referenced from different data sources and/or applications. In the tree view the following different screen-related icons are used:



- references to list screens (a.k.a. brief display screens)



- incorrect reference (screen file not present)



- references to detailed display screens



- incorrect reference (screen file not present)



- references to search screens



- incorrect reference (screen file not present)



- screen files.

Screen files in the Screens manager

1. Start the *Screens manager* from the main Designer window by choosing *View > Screens manager* or by clicking the button for it:



2. In the *Screens manager*, click the *Change working folder* button to select a work (sub)folder that contains screen files:



(Click here for information about how to find a certain screen.)

3. The screens occurring in the selected folder will be presented in a list containing the screen file names, their descriptions, markers to indicate whether the screen has been changed and needs saving, and the last-modified date for each screen. Double-click a screen name to open it in the *Screen editor*.

See also

Managing screens

Managing screen objects

Editing screen object properties

Saving modifications

2.3.2 Managing screens

In Adlib Designer there are two ways to manage screens: through the *Application browser*, or through the *Screens manager* (see *Accessing screens*). In both ways you are able to create new screen files, delete or save screens, move or copy them to other folders, and open them for editing in the *Screen editor*.

Where to find a particular screen

Screen files may be located in different subfolders of your Adlib software folder. Typically you can find them in `\screens`, `\data\zoom`, or folders with an application name like `\wincat`, `\library`, `\museum plus`, etc.

If you are searching for a particular screen that you saw in your running application, you can look up its name in that running

application by opening the concerning screen, for instance by clicking the concerning tab in the detailed display of a record, and pressing **Ctrl+Alt+S** on your keyboard. In the top of the window that opens, the relative path and file name of this screen is displayed. If there is no relative path, it means that this file can be found in the current application folder.

Find in application tree

To search for a screen name in the tree view of the *Application browser*, choose *Edit > Find* (**Ctrl+F**) or click the button for it:



In the *Search for* entry field, type any name or part thereof that you want to search in all of the text displayed in the *Application browser* tree view.

If the term you type is only part of a word or words you look for, then deselect the *Match words* option.

Leave the *Match case* option unmarked if upper and lower case are not important while searching.

Click *Find* to start the search. A found term is highlighted. To search a next appearance of the searched term, each time click *Find next*, press **F3** or click the button for it:



Finding screen references

A screen file may be referenced more than once in your Adlib applications. To find all these references, before you start editing a screen for example, you could use the Designer *Object searcher*. (Search the *Object category: Screen*, for the name of the screen file, e.g. `booka`, in the *File specification: Applications*.)

Sorting a screens list

When you already know in what folder to look for a screen, you may still not quickly find that screen because the list is long and unsorted. Sorting the list makes finding your file a lot easier.

The files under a folder node in the *Application browser* (not the objects in an application definition) can be sorted alphabetically. Just right-click a folder node and choose *Sort* in the pop-up menu. You can only sort on the descriptive names here, not the file names, but the file names are displayed too.

In the *Screen manager* on the other hand, you have more sort options: you can sort on the screen file names, the descriptive names, the needs-saving markers, or the last-modified dates, by clicking the appropriate column header once or twice (for an ascending or descending sort).

Copying and moving

In the *Application browser* you can move or copy screen files or screen references to another location, but only one at a time (or you could try copying an entire folder). Moving can be done by cutting and pasting or by simply dragging the object.

Right-click an object and choose *Copy* (**Ctrl+D**) or *Cut* (**Ctrl+Y**) in the pop-up menu. Then select another list or folder, right-click it and choose *Paste* (**Ctrl+I**) in the pop-up menu.

Screen references should only be copied or moved to other lists of screen references, while screen files can be copied or moved to any folder node.

When you paste an object that is part of an application definition, it will never overwrite the other selected object or another object with the same name in that list.

As mentioned, you can also drag objects from one list or folder to another (to where the mouse pointer displays a +), or in the same list. (Dragging means clicking an object, keeping the left mouse button pressed down, and moving the object some place else, and then releasing the mouse button.) Dragging a screen file to another list or folder means copying it. Dragging an application object to another place in the same list, means moving it; this is relevant for the data sources list a/o, because the order of this list is the order in which it is presented to the user in the running application.

In the *Screens manager*, copying and moving is somewhat more clear-cut. First of all, you can select one or more screen files simultaneously: either hold the **Ctrl** key down as you click each next file to be selected, or hold down the **Shift** key as you click the last file to be selected if you want all files between the previously selected file and the last to be selected at once. Then right-click the selection, or open the *Edit* menu, and choose *Cut* (**Ctrl+X**) or *Copy* (**Ctrl+C**). Then select another folder, and paste the selection through the *Edit* menu.

Creating screen files and screen references

To create new screen files in the *Screens manager* you must first open the folder in which you want the new file to be inserted. Then just click the *Create a new screen* button:



When you create a new screen, you will be asked for the name that the file should get. Enter the name for the new screen, without extension. This is the name of the physical (*.fmt*) file, not the title of the screen. In DOS environments (ADSETUP) the name may be only 8 characters long; Windows supports long file names. After you've provided a name, the file is created and you can directly start editing it.

In the tree structure in the *Application browser* you can create new screen files and screen references.

Select the folder in which you want to create a new screen file, and create the object through *File > New*, or right-click the folder and choose the new object through the *New* option in the pop-up menu. In the *Application browser*, new screen files can only be created from a folder node.

What objects are available in the *New* menu depends on the node that you selected.

From a data source node (or the *Screens* node underneath it) or a method node in an application definition, you can create references to existing screens: just right-click the node to which you want to add a screen reference and choose the type of reference you wish to add (this is because Adlib cannot tell from the screen file itself, what its intended use is, for detailed display, brief display, or as search screen).

In the *Application browser* there is a handy feature for creating screen references to existing screen files, wherever those files might be located.

1. First open the appropriate application definition node so that the data source or method to which you want to add a screen reference is displayed in the tree view.
2. Then also open the folder that holds the screen file to which you want to reference.
3. Click the desired screen file and drag it to the destination data source or method. The tree view will automatically scroll up or down as needed.
4. When you release the mouse button a small dialog pops up, asking you which type of reference you wish to create. Choose the type corresponding with the dragged screen file, and the correct screen reference is then created for you. Automatically the proper relative path (if any) to the screen file will be retrieved, for the screen reference property.

Opening screens in the Screen editor

You can open more than one screen in the editor, either by switching back to the *Screens manager* or *Application browser* (without closing the *Screen editor*) and double-clicking another screen, or by opening a new screen file in the editor. Each screen is presented on its own tab, and you can switch between them by clicking the tab labels.

Note that you can also start a new screen from within the *Screen editor*. It will be created in the currently opened folder.



Just click the *Create a new screen* button, and then enter the name for the new file, without extension. Screen file names are no longer limited to 8 characters. (But note that if you use longer names, you won't be able to view those names in full in the old DOS tools anymore.)

Deleting screens

In the *Application browser* or in the *Screens manager*, select a screen file that you want to delete - in the *Screens manager* you can select more than one file - and either choose *Edit > Delete*, or right-click the object and choose *Delete* in the pop-up menu, or click the *Delete* button:



Deletion of files is currently permanent: you cannot restore deleted files from Windows' recycle bin. Deleted sub-objects within the application definition (like screen references), can be restored by not saving the changes in the concerning *.pbk* file when you close Designer.

See also

Accessing screens

Managing screen objects

Editing screen object properties

Saving modifications

2.3.3 Managing screen objects

The *Screen editor* opens when you double-click an existing screen or when you start a new screen file, from the *Screens manager* or the *Application browser* (see *Accessing screens*). The *Screen editor* allows you to edit the layout of the screen and the properties of all objects on that screen and of the screen itself. The screen will be laid out as it would in a running Adlib application. The possibilities are as follows:

Setting the interface language

In the *Language* menu you can change the language in which you view and edit the screens. Make sure you select the proper language first, when you start editing a screen, so that the displayed texts on screen are always in the correct language.

When you place for instance a new label on the screen, you are allowed to type the text for it immediately in the physical control. But if your application supports different languages you must type this text in the currently set application language. So set the language to English if you want to enter English texts and let all other texts on the Adlib screen also be displayed in English. And for instance change the language to Dutch to view or enter Dutch texts on screen. The advantage of translating this way is that you can see directly if a text fits the space for it (with this font anyway). To ease translating like this, the status bar holds the default (English) name of the field or label that is currently selected, so you don't have to switch back and forth between languages. But if you tend to forget to set the right language, or just want to enter all language texts per control at once, it may be easier to translate such texts in the properties of the concerning screen object; in there you can enter all language texts at once, without having to switch the application language.

Inserting new screen objects

Through the *Insert* menu you can add new objects to a screen. By default these are placed at the bottom of the screen.

You can also right-click the screen or an object, and choose *New* in the pop-up menu to insert a new object directly under the mouse pointer, instead of at the bottom of the screen*.

You can add the following objects:

- *Fields* - In an entry field the user can read the value from the associated database field, or enter a new value. A new entry field will automatically come with an associated label to the left of it; it's important to keep them together, although the label-part can be deleted separately if you don't want the entry field to have a label at all. (When you delete an entry field, the accompanying label will be deleted too.)

- *Labels* - Separate labels are used to display a fixed text on the screen, without it having to do with any specific entry field. Such labels are typically only used in recent brief display screens (to hold separator texts). See the Help topic: Designing different screen types, for more information.
- *Boxes* - Boxes are cosmetic frames to visually group entry fields that sort of belong together. A box comes with an integrated label in the top left corner.
- *Images* - If you want the Media Viewer (see the Adlib User guide) to appear on this screen by default (if this is a detailed display screen) to display the referred to image(s) from a database field, or (if this is a brief display screen) to display a thumbnail image next to each listed record, then place an *Image* object on your screen.
- *Menu options* - A menu option (a.k.a. checkbox) is used in detailed display screens if a field can only be on or off, or be yes or no, and in brief display screens to mark records.
- *System fields* - A system field is used on brief display screens to concatenate all data and fixed text that must be displayed to briefly list a record. Which data and text, is determined on the *List fields* tab of the screen properties.
- *Text windows* - A text window places a small window on screens in the detailed presentation of a record to display different fields next to and beneath each other, as a sort of rudimentary print preview.
- *Buttons* - <not fully implemented yet>
- *Web browser controls* - A web browser control displays data from the current record as an HTML page in a fixed box on the screen, by means of an XSLT stylesheet which must transform the XML format of the Adlib data to HTML.
- *HTML fields* - An HTML field is a field meant for long, laid-out text. Layout can be applied to the text during editing of the record. You can print the contents of such a field to a Word template or with the aid of an XSLT stylesheet, whilst keeping the layout intact. Although you will see just the laid-out text while you are editing an HTML field, the field contents will actually be stored as HTML code in the background.
- *Empty lines (Shift+Insert)* - When you edit a screen with a lot of objects on it, it is not so easy to manually create a empty line somewhere in between, because you have to select and move many objects. Instead, you can insert an empty line; select an

object above which you want the empty line to appear, insert the empty line, and note that all objects below it have moved down and that the current box has been resized too.

* Note that if you insert objects from the pop-up menu, not all object types may be present in the pop-up menu. This is because some objects you cannot insert into certain other objects. In a field for example, you can't insert any object at all.

Selecting and deselecting screen objects

To move, align, copy or cut one or more objects on the screen, you have to select them first.

You can select a single object, by simply clicking it. A selected object is displayed differently, its border line being doubled or four little black squares being placed around its corners.

You can select more than one object, by holding the `ctrl` key down while clicking all objects that you want to select.

If you select an entry field, you automatically also select the associated label (if it has one).

You can deselect all selected objects at once, by clicking an empty part of the screen. Note that the space inside a box is NOT an empty part of the screen: if you click the space inside a box, you select the box.

You can deselect one object in a group of selected objects by `ctrl`-clicking it again.

Moving, copying, cutting or deleting a selection of objects

You can move an object by clicking it, keeping the mouse button pressed down, and dragging it to another position. A group of selected objects can be moved together by dragging one of the selected objects in it.

Moving snaps to a grid of 80x25 cells, each 10x25 pixels in size. You cannot move or size outside of these dimensions. This is done to maintain compatibility with the existing Adlib software and the DOS based tools ADSETUP and DBSETUP.

Screen objects may be partially or completely hidden by other objects. When you are working with conditional fields it may be desirable to place entry fields on top of each other, but mostly you don't want this. To change the "stacking" order in which these objects appear (to be able to move them elsewhere) you can bring a selected control to the front of the stack (make it visible) by clicking the *Bring to front* button:



or send it to the back with the *Send to back* button:



You can copy or cut one or more selected objects, either by right-clicking one of them and choosing *Copy* or *Cut* from the pop-up menu, by choosing these functions in the *Edit* menu, or through the shortcuts **Ctrl+C** or **Ctrl+X**.

You can paste the copied or cut object(s) in the same screen or in any other opened screen, by right-clicking the location where you want to place them, and choosing *Paste* in the pop-up menu.

You can remove a selection of one or more objects, by clicking the *Delete selection* button:



There are some keyboard shortcuts available for navigating and moving screen elements:

Key or shortcut	Function
 , Shift+Tab	Selects the object to the left, or above.
 , Tab	Selects the object to the right, or below.
Home	Selects the first object, in the left upper corner, of the screen (not a box).
End	Selects the last object, in the right lower corner, of the screen (not a box).
Shift+  ,  ,  , 	Changes the size of the currently selected object(s) in the direction of the arrow, if the object allows it (see the next paragraph).
Ctrl+  ,  ,  , 	Moves the selected object(s) in the direction of the arrow, but won't move other objects accordingly, so overlapping may take place.
Shift+Insert	Inserts an empty line above a selected object, and moves all objects below it one line down.
Shift+Delete	Deletes the line that holds the currently selected screen object, and moves all objects below it one line up.

Resizing screen objects manually

The size of some screen elements can be changed manually, some can't, and others are set automatically. If adjustable, length can be changed in steps of 10 pixels (equals 1 column), and height in steps of 25 pixels (equals 1 row). Numerical coordinates (measured from the top left corner of the screen) and dimensions of screen elements are displayed in the status bar of the *Screen editor*, in two possible formats:

- *(row, column)-(row, column)* for image objects, boxes and text windows, indicating the location of the top left corner of the selected object followed by the location of the lower right corner of the same object.
- *(row, column)[length]* for fields, labels, system fields and menu options, indicating the location of the top left corner of the selected object followed by the length in number of columns.

In general, you can resize a screen object, if a double arrow appears when you move the mouse pointer over a border of the element (usually the right and/or lower border). More specifically:

- **Fields, System fields** - Select a field, move the mouse pointer over the right border until it turns into a double arrow. Now click, hold the mouse button down and drag the field longer or shorter. Only on this side of the field you can adjust its length, you can't adjust the height.
Note that the actual data content can be longer than the entry field, but this content can never be longer than the associated data dictionary field length defined in the database setup; if a database field contains more text than can be displayed in the entry field, then place the cursor in it and move to the right with the arrow key, to scroll through the contents.
- **Labels** - Labels are automatically sized to fit the space in between the left screen border or the right border of a field to the left of it, and the right screen border or the left border of a field to the right of it. Click anywhere in a (selected) label and drag it left or right to move it, and thereby expand or shorten its length, whilst keeping its right border at the same location.
- **Boxes, Images, Text windows** - You can resize these screen objects by dragging their right or lower borders.
- **Menu options** - Menu options cannot be resized.

Aligning or resizing screen objects semi-automatically

You can align or resize all selected screen objects to the LAST

selected element. **Ctrl**-click the desired elements, and remember to select the proper last element. Then click the desired align or resize button in the toolbar:

Button	Name	Function
	<i>Align left</i>	All selected objects are moved left or right so that their left border is aligned with the left border of the last selected object; their length remains unchanged.
	<i>Align right</i>	All selected objects are moved left or right so that their right border is aligned with the right border of the last selected object; their length remains unchanged.
	<i>Align top</i>	All selected objects are moved up or down so that their upper border is aligned with the upper border of the last selected object; their height remains unchanged.
	<i>Align bottom</i>	All selected objects are moved up or down so that their lower border is aligned with the lower border of the last selected object; their height remains unchanged.
	<i>Make same width</i>	The length of all selected objects is changed (if possible) to the length of the last selected object; the position of their left upper corner remains unchanged.
	<i>Make same height</i>	The height of all selected objects is changed (if possible) to the height of the last selected object; the position of their left upper corner remains unchanged.

Connecting or disconnecting fields and labels

Currently, you cannot disconnect a field from its accompanying label (other than deleting the label), or connect a separately inserted label to a field without a label.

Opening the properties of a screen object

The properties of any screen object can be opened by right-clicking the object and choosing *Properties* in the pop-up menu. Some objects only have one property, others have multiple, spread out over several tabs. You can open the properties of the screen itself by right-clicking an empty space on the screen (not somewhere in a box) and also choosing *Properties* from the pop-up menu.

Note that the caption property of labels and boxes, and the tag property of fields, can also be edited in the visual objects themselves, instead of in their properties.

See also

Accessing screens

Managing screens

Editing screen object properties

Saving modifications

2.3.4 Editing screen object properties

The properties of any screen object can be opened by right-clicking the object and choosing *Properties* in the pop-up menu. Some objects only have one property, others have multiple, spread out over several tabs. You can open the properties of the screen itself by right-clicking an empty space on the screen (not somewhere in a box) and also choosing *Properties* from the pop-up menu.

Note that the caption property of labels and boxes, and the tag property of fields, can also be edited in the visual objects themselves, instead of in their properties. (Then do remember to set the correct interface language.)

Moving through properties

You can easily move downwards through properties by pressing **Tab** on your keyboard, or **Shift-Tab** to move upwards.

Typing property values

If properties of an object are displayed in white or yellow table cells or entry fields, you can edit them. (The yellow colour is just for visual presentation, it has no special meaning.)

	Language	Text
▶	English	Abstract and subject terms
	Dutch	Samenvatting, classificatie, trefwoorden
	French	Résumé et termes sujets
	German	Abstract und Schlagwörter
	Custom1	
	Custom2	
	Custom3	Αφηρημένοι και θεματικοί όροι

In a table cell or property entry field, just delete or type characters.

Regular expression

If properties of an object are displayed in greyed out entry fields, you cannot edit them: they are read-only.

Sometimes you can only choose values from a drop-down list. If such a property reads "*Undefined*", it means that for that property the default value will be used: usually the default value is the first value in the drop-down list.

Choosing a value from a list

Zoom screen

Behind some property entry fields, you'll see a grey button with three dots on it. Click this button to open a list of files or items that are available for the current property, to search your system for a specific file name or item. Whenever the button is available, you are advised to use it: choosing an object from the list makes sure its name is spelled just the right way and that any path to the object is entered correctly.

Choosing colours

For several objects you can set a foreground colour and/or background colour.

Foreground Data example

Background

Just click the coloured box that you want to set differently, to open a standard Windows colour picker from which you can choose a new colour. To the right of the coloured boxes, you'll sometimes see an example of how the two colours look together.

You can assign colours to individual objects, to all objects on a

screen, or even to all screens in a certain folder; click here for more information about colouring your application efficiently.

See also

Accessing screens

Managing screens

Managing screen objects

Saving modifications

2.3.5 Saving modifications

Screens that you have edited in the *Screen editor*, can be saved directly from the *Screen editor*, from the *Screens manager*, from the main Designer window, or when closing Designer.

From the Screen editor

Changes* in the layout of a single screen or in the properties of a screen and the objects on it, can be saved directly manually in the currently visible screen (*.fmt* file) by choosing *Save* (Ctrl+S) in the *File* menu or by clicking the *Save current screen* button:



Choose *Save as* in the *File* menu if you want to save the screen file under a different name than it currently has; the old screen file will not be deleted.

Note that when you create a new screen, it is immediately saved under the name you provide. If you decide later that you won't be needing this screen anymore, you will have to delete it from your system manually, via the *Application browser* or Windows Explorer.

From the Screens manager

If you've closed the *Screen editor* without saving the edited screens you can still save them here, or from the main Designer window. In the *Needs saving* column in the screens list in the *Screens manager*, any edited screens are marked *Yes*. Of these marked screen files you can choose which ones to save and which not.

If you select only one file, you can save it by choosing *Save* (Ctrl+S) in the *File* menu.

You can also select more than one file by holding the *ctrl* key down

and clicking all files to be selected. Then click the *Save selected screens* button:



If you just want all edited screen to be saved, you could also choose *Save all* (**Ctrl+Shift+S**) in the *File* menu.

From the main Designer window

In the main Designer window or in the *Application browser*, you can also save changes in edited files (not only screen files), by choosing *File > Save all...* or by clicking the *Save all* button:



In the *Save objects* window that opens when you click this button, you will be presented with an overview of all the unsaved files in which you made changes, and you can determine of each of these separately whether you want to save them or not, by selecting them or deselecting them. Click *Yes* to save the selected files. *Cancel* returns you to Designer.

When you close Adlib Designer while you haven't saved all changes yet, the *Save objects* window will automatically appear, allowing you to save your work before the application is closed.

* The nature of properties forms and lists sometimes requires you to leave a property that you just edited, before you can save it. So when you have edited your last property for today, make sure you first click some other property or tab, before you save all your work.

2.4 Properties of screen objects

§

2.4.1 Screens

2.4.1.1 Screen

After you have opened a new or existing screen in the *Screen editor*, right-clicked an empty part of that screen and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Screen* tab in that window, you determine the general properties of this screen.

Click here for information on how to edit screen object properties in general. And click here to read about how to manage screens in the tree view in the *Application browser* or in the *Screens manager*. On the current tab you'll find the following settings:

Name

The screen name is the name under which the file is referenced to internally in Designer. This name should be the same as the file name under which this screen has been created and saved, but must be written without extension (*.fmt*).

The name should be a concise indication of the purpose of the screen and its type. In Adlib applications for Windows, screens usually have a type indication as follows: brief display screen names start with *BR_* (e.g. *BR_COPY*), link screens start with *Lnk*, Query by form screens with *QBF*, zoom screens with *ZM_*. It may also be handy to include a data source name abbreviation in the screen name to indicate from where the data for this screen must be coming.

In DOS environments this name can consist of only 8 characters; in Windows you can use long names.

Screen descriptions

You must provide a brief description of the screen in one or as many languages as you wish: these descriptions will appear on the label of the screen in the running application.

Width and Height

The screen width and height set here, are the width and height of the screen in pixels. By default this is 800x625. A blue line at the right side of the screen in design mode indicates the 800-pixel border. (This line serves only as a visual aid, it won't be visible in the running application.) If you define a screen bigger than the screen resolution of the monitor that will display it later on in the finished application, or bigger than the user's current Adlib application window size allows it to be, then scroll bars will appear automatically to make all of the screen accessible for editing and viewing records.

If you don't want users to have to do a lot of scrolling, then the default screen size is usually a good choice.

Read-only

Leave this option unmarked to allow users to start editing or delete a currently opened record from this screen in detailed display mode, or the currently selected record from this brief display screen.

Typically you leave this option unmarked, because if you want to restrict write access to records for certain groups of people, you

should use other access mechanisms, especially since limiting access rights applied through the other available mechanisms always have priority over this edit/read-only setting. So with this option unmarked, all users can edit the current record from this screen, unless limiting access rights apply to those users.

Previously, this functionality was achieved by the EDIT parameter that was placed on a screen like fields etc. This parameter is not visible on the screen in design mode in Designer; instead, the *Read-only* property is left unmarked. Marking this option equals removing the EDIT parameter from the screen.

Show empty fields

Mark the *Show empty fields* option to instruct Adlib to display all fields (actually, their linked labels) on this screen in the detailed display mode of a record, even when certain fields do not contain any data. For most applications this is preferred. Note that this option should only be set for screens that are used in the detailed display of records and for zoom screens.

Previously, this functionality was achieved by the DISPALL parameter that was placed on a screen like fields etc. This parameter is not visible on the screen in design mode in Designer; instead, the *Show empty fields* property is marked. Deselecting this option equals removing the DISPALL parameter from the screen.

Help key

This is the name of the Adlib Help topic corresponding to the current screen. If the user of the running application views this screen and presses F1 (Help), the user will see the Help text associated with this key.

Before screen adapl

Enter or search for the name of the adapl that you want Adlib to run before a record or list of records is displayed on screen. An example of such an adapl is a procedure for a brief display screen which takes an author's name (like: Austen, Jane) and places the first name in front of the last name, and deletes the comma behind the last name. More complex concatenation and editing of strings is of course also possible; the result is usually put in a temporary tag, purely for display on the screen.

Note that a *Before screen adapl* should only be set for brief display and detailed display screens.

After screen adapl

Enter or search for the name of the *adapl* that you want Adlib to run whenever a user leaves this screen in a running application: this happens for instance when the user switches to a different tab or just before an edited record is saved from this screen.

Note that an *After screen adapl* should only be set for detailed display screens. (An *After screen adapl* is never called after a brief presentation screen.)

Object colours

For the currently opened screen you can set foreground and background colours for all labels and fields on it*, all at once.

At the bottom of the current properties tab you'll see the *Foreground colour* and *Background colour* options for *Labels*, *Data* (fields and their contents) and *Highlighted data* (a selected record on this screen for brief display - so this doesn't apply to detail screens), and below them, the *Screen background colour* option.

The foreground colour is always the text colour, while the background colour concerns the colour behind that text. Click the coloured box that you want to set differently, to open a standard Windows colour picker. You can select a basic colour, or click *Define Custom Colours* to choose another colour.

For the currently opened screen you may also set a background colour. This is the colour that will be visible around the fields and their labels (boxes are currently always transparent). Click the coloured box next to *Screen background colour* to open the standard Windows colour picker, and choose the desired colour.

If you would like to apply a certain set of colours to more than one screen and the elements on them, you'll have to work with colour schemes. Click here for more information about using colour schemes.

* Note that the colours that you have assigned to individual labels or fields won't be overwritten if you assign colours to all labels and fields at once, like described above.

2.4.1.2 List fields

After you have opened a new or existing screen in the *Screen editor*, right-clicked an empty part of that screen and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *List fields* tab in that window, you may list tags, parameters and separator texts that build up this screen if it is a brief display screen. For a brief display screen only, the options on this tab have any use. But a brief display screen can be constructed in different ways, and only one of those ways makes use of the *List fields* tab, so often the

options on this tab will be left empty.

Click here for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

Tag

To this list you may add database tags in the order in which they must appear on the currently defined brief display screen: the top tag is the first tag and will be displayed on the left side of the brief display screen in a running application.

Use the buttons to the right of this list to edit its contents and/or its sorting order.

You may also add applicable Adlib parameters to this list. Since the old EDIT parameter is now present as the unmarked *Read-only* option on the first screen properties tab, you must not enter or search it here. It depends on the operating system under which you run your Adlib application, what parameters you can use here. But for a brief display screen to be used in an Adlib Windows application there is only the special parameter: **, to indicate a field separator. In the *Tag* list you typically enter the double asterisk in between database tags; the character(s) with which you want to separate the concerning tags, must be entered in the *Separator texts* property to the right of the buttons.

For example, the *Brief display copies* screen (*br_copy.fmt*) in a Library 2.1.1 application has the following *Tag* list:

e0

**

e3

**

e4

Separator texts (Language, Text)

Select a tag or parameter in the *Tag* list to the left, to view or edit the separator text for this tag or parameter, in different languages. If for a language no text has been provided, by default the English text is used instead (if available).

A separator text for a tag indicates the text with which any occurrences of this tag will be separated, while a separator text for the ** parameter indicates the text with which the tags will be separated. An often used separator text for ** is: /, preceded and followed by a space.

Note that you can use spaces in separator texts normally. the ~

character that was used behind trailing spaces in a separator text in the old ADSETUP tool, needs no longer to be used in Designer. In ADSETUP created screens that used this character, won't display it when opened in Designer, while the spaces remain present. If you enter the ~ character in a separator text on a screen opened in Designer, the character will be interpreted as any other character and displayed literally on this brief display screen in a running application.

2.4.1.3 Screen conditions

After you have opened a new or existing screen in the *Screen editor*, right-clicked an empty part of that screen and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Screen conditions* tab in that window, you can set conditions under which the current screen must be hidden from view.

Click [here](#) for information on how to edit screen object properties in general. And click [here](#) to read about how to manage screens in the tree view in the *Application browser* or in the *Screens manager*. On the current tab you'll find the following settings:

Conditions

Enter a single expression or a plural, Boolean expression as the condition to show the current screen. Every time you open a record for display or editing, and every time you leave a field in edit mode, all relevant expressions will be evaluated. When this screen is associated with the current data source and if the screen condition is true, the screen will be shown in the detailed display of a record, while if the result is false, the screen will be hidden. See the general topic *Conditional screens and fields* for a full description of this functionality and its possibilities.

Suppressing screens this way is not a safe way to protect confidential information, because there would still be other ways to extract the information you want to protect. Confidential information is better protected with the proper access rights.

The basic form of a single expression is:

<field or constant> <operator> <value>

Examples:

```
material = 'book'
```

Result: the screen will be displayed if the *material* field contains the value *book*.

```
name.type = 'SUPPLIER'
```

Result: the screen will be displayed if the enumerative *name.type* field contains the neutral value `SUPPLIER`.

```
$preref > 1000
```

Result: the screen will be displayed if the record number of the current record is greater than 1000.

```
$newrec
```

Result: the screen will be displayed if the record you just opened is a new record.

```
$role != student
```

Result: the screen will be displayed if the role of the current user is unequal to `student`. `$role` returns an empty string if for the current user no role has been specified, in which case the result of this expression is true.

```
address.postal_code ~ '[1-9][0-9][0-9][0-9][A-Z][A-Z]'
```

Result: the screen will be displayed if the postal code field contains a postal code which satisfies the regular expression.

```
$preref > 1000 && $role = financial
```

Result: the screen will be displayed if the record number of the current record is greater than 1000, and the role of the current user is `financial`.

```
material = 'book' || material = 'serial'
```

Result: the screen will be displayed if the *material* field contains either the string value `book` or the value `serial`.

```
$newrec && (material = 'book' || material = 'serial')
```

Result: the screen will be displayed if the material field contains either the string value `book` or the value `serial`, while this record is new.

2.4.1.4 Access

After you have opened a new or existing screen in the *Screen editor*, right-clicked an empty part of that screen and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Access* tab in that window, you determine the access rights for this screen to restrict access to it, dependent on the user (login name in Windows) and its assigned role.

Click here for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

Access (Role, Access rights)

Here you may define which *Roles* have which *Access rights* to this screen. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. No access means the screen won't be visible at all; *Read* access allows the user to view data on the screen but not to edit it; *Write* and *Full* access allow the user to edit the data on this screen.

None and *Read* have priority over the unmarked *Read-only* option for a screen, meaning that the *Read-only* option only makes a screen editable in principle, but that you can still exclude certain users from doing that, through access rights. These two access rights also have priority over less limiting access rights set on other levels in the application or databases.

If a role is not linked to this screen, then each user linked to that role has full access by default, unless restricted by other limiting access rights. A user without a role always has full access.

Users are assigned to roles, in the application setup.

Note that setting access rights on screen level is rarely put to use, since there are better ways to prevent certain data from being edited or viewed, for instance by using application identification roles, or setting access rights on database field level (see the reference below for more information).

See also

Security in Adlib

2.4.2 Screen fields

2.4.2.1 Data

After you have opened a screen in the *Screen editor*, right-clicked a screen field (a.k.a. entry field) on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Data* tab in that window, you determine the general properties of this particular screen field, like its associated database tag and some interface settings. A screen field is necessary to display the contents of a database field on screen, and to allow the user to enter or edit data in that database field.

Click here for information on how to edit screen object properties in general. And click here to read about how to manage screen fields in the *Screen editor*.

On the current tab you'll find the following settings:

Tag

Fill in the existing data dictionary tag for this screen field. However, if you don't want to have a data dictionary definition for the database tag you fill in here, e.g. because it is a temporary filled screen field, concatenated from other fields, then it's also possible to fill in a tag that doesn't exist in the data dictionary.

Note that tags are case sensitive.

Data type

For the above entered tag, choose the data type that corresponds to the way that field is defined in the data dictionary, and which determines what you can enter in this field. Possible data types are:

Data type	Meaning
<i>Text</i>	Will accept all characters.
<i>Letters</i>	Will only accept alphabetic characters and spaces.
<i>Numeric</i>	Will only accept numeric characters. Use a full-stop as the decimal symbol. The numeric value may be preceded by a plus or minus sign.
<i>Integer</i>	Will accept only whole numbers. The numeric value may be preceded by a plus or minus sign.
<i>Date</i>	Will accept a date. The following notations are allowed:
	<i>dd/mm/yy</i> (e.g. 31/12/94)
	<i>dd/mm/yyyy</i> (e.g. 31/12/1994)
	<i>yy-mm-dd</i> (e.g. 94-12-31)
	<i>yyyy-mm-dd</i> (e.g. 1994-12-31)
	<i>yyyy-ddd</i> (e.g. 1994-365)
	The dd-mm-yyyy is still accepted, but is now deprecated.
<i>Time</i>	Will accept a time. The only accepted notation is <i>hh:mm</i> (e.g. 23:55)

<i>ISBN</i>	A valid ISBN (number), either with or without punctuation (e.g. 90-03-90201-1 or 978-90-03-90201-6)
<i>ISSN</i>	A valid ISSN (number), either with or without punctuation (e.g. 0040-9170)

Repeatable

Choose whether this field may be *Repeated*, *Not Repeatable*, or *Repeatable*, *values must be unique*. Repeated screen fields can have more than one occurrence, whilst *Not Repeatable* can only have one. With *Repeated*, *value must be unique* every occurrence must be different from the others for the same field.

When you were to add subsequent repetitions of a field during design (which is rare), this property will be set to *Not Repeatable*, regardless of what you have set for the first occurrence of the field.

This is because when a field occurs more than once in a screen, Adlib automatically detects that it is a repeated field. It is therefore no longer relevant to set this option. You only need to set this option when there is only one occurrence of a field on a screen and you want to indicate that it is a repeatable field.

If you have set the length of a repeatable field to 0 in the data dictionary, Adlib will automatically use word wrapping. This means that Adlib automatically sends a word to the next line of a screen field (starting a new line if necessary) if the word no longer fits. The reverse happens if you delete text. Adlib always reorganises the text to make it fit the entry field as well as possible. Of course, Adlib will never change the order of the text.

Justification

Choose *Left*, *Right*, or *Center* to specify how data in this screen field should be aligned (meaning: to position field data within the horizontal space of the screen field). The *Left*, *Right* and *Center* options speak for themselves. The *None* option displays the data exactly as it has been entered in the database, including any preceding spaces. The default setting is *Left*.

Edit access

When the screen itself is editable, you can still specify for individual entry fields that they are not editable, by choosing *ReadOnly* for this property.

Note that it is never possible for a user to change the value of the %

0 (record number) tag, even if this property is set to *ReadWrite*. Similarly, a user will be unable to make changes if he or she does not have write access in the database, or if the screen is not editable.

Data validation

Choose *None* (default), *Not empty*, *Complete*, *Mandatory group*, *Before edit adapl*, or *After edit adapl* to specify what type of input checks Adlib should use when data is entered in the current screen field:

Method	Action
<i>None</i>	Adlib does not check whether the field has been filled in or not.
<i>Field is mandatory</i>	Adlib checks that some data has been entered in the field, but does not check the actual data.
<i>Field must be complete</i>	Adlib checks if the number of entered characters is equal to the length of the associated data dictionary field.
<i>Field is mandatory in group</i>	<p>If the current field is part of a group, then Adlib checks that if at least one other field within this group has been filled in, that the current field is filled in too. If no other field in the group has been filled in, then the current field needn't be filled in either.</p> <p>A group may be defined either on screen level or on data dictionary level. If both types of group definition exist for this field, and differ, then the data dictionary group has priority. Adlib advises to only use data dictionary groups.</p>
<i>Run adapl before entering this field</i>	<p>Adlib will use the <i>Field based procedure</i> set for the current database, when the current screen field becomes active (e.g. when the cursor is placed in the entry field). If no <i>Field based procedure</i> has been set, Adlib will use the <i>Before screen adapl</i> which may have been set for the current screen.</p> <p>The adapl may for instance check the data entered in other fields and/or maybe fill the current field with some value.</p>

Run adapl after leaving this field

Adlib will use the *Field based procedure* set for the current database, when the current screen field is left (e.g. when the cursor is placed in another entry field, or when the user switches tabs). If no *Field based procedure* has been set, Adlib will use the *After screen adapl* which may have been set for the current screen.

The adapl may for instance check the data entered in this field to a number of criteria. Note that if the current screen field is associated with a linked field, and the user just entered a different existing linked term in the field and leaves it, then the after-field adapl will be executed before the link has been resolved: this means that when the adapl runs, no link reference or merged-in values are available yet, just the linked value itself.

Regular expression

A regular expression is sort of a template that prescribes how the contents of a field should be formatted. You can use it to indicate what characters Adlib should accept in the input and what characters should appear at what position. [Click here](#) for the full Help topic about regular expressions.

If the current screen field is filled in by the user, it will be validated to any regular expression you provide here, when the field is left. You can use this functionality if user input in a certain field must adhere to particular rules, like an object number that must always start with two alphabetic characters, followed by exactly six numbers for instance.

Foreground & Background colour

Set a foreground colour and/or background colour for the current field only, if you want to set it apart from most other fields on the screen. This way you can stress the importance of certain fields, for instance. Colours that you've set for individual labels and fields, will normally not be overwritten by colours that you set on any other level.

Click the coloured box that you want to set differently, to open a standard Windows colour picker from which you can choose a new colour. To the right of the coloured boxes, you'll see an example of how the two colours look together.

If you would like to assign colours to all (or most) fields on this screen, or even to all screens at once, you should not use this option; click here for more information about colouring your application efficiently.

2.4.2.2 Advanced

After you have opened a screen in the *Screen editor*, right-clicked a screen field (a.k.a. entry field) on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Advanced* tab in that window, you'll find some practically redundant and/or rarely used settings.

Click here for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

Occurrence

For most purposes this property should be *1*. If the field is repeated, all occurrences will be displayed anyway.

Field group

If you want several fields to be treated as a group, when adding or displaying occurrences (so that all grouped fields get a new occurrence at the same time), you may assign to these fields a (screen) field group number, other than zero. This group number has no significance outside this screen. You can create as many screen field groups as you like by assigning each one a different number.

Note that you can define a group **name** on data dictionary level in the database setup or a group **number** on screen level when editing the properties of a field on a screen. If you want groups of fields on a screen also to be accessible as a group from within ADAPL or Internet Server, you should define the group in the database setup only; setting up the same group on screen level too may cause conflicts. Data dictionary groups do usually have priority over screen level groups but you can't take this for granted. It's possible to define groups only on a screen, but their use is limited to the application that incorporates that screen and *adlwin.exe* search-and-replace actions in screen field groups can only replace values in the first occurrence of the field group. Therefore it is recommended to only specify data dictionary field groups.

Zoom screen

Select the screen in which Adlib must display (and only for display, not

to edit) details of a linked record from a secondary data source, when the user clicks the currently specified underlined linked field in a displayed record in the primary data source, or when the user clicks the *Show* button for a selected record in the list in the *Linked record search screen* for the currently edited linked field.

There is no default zoom screen for these situations, so the user cannot 'zoom' in a linked field without a zoom screen specified for it. You can also specify a zoom screen for a linked field in the database setup, but a zoom screen specified for a screen field has priority over any zoom screen specified in the database setup. In most cases, zoom screens are associated with linked fields in the database setup, because the data dictionary field is only defined once, while screen fields may appear on several screens and in different applications, but usually need the same zoom screen anyway.

A relative path to a screen must be relative to the application folder, so for instance: `..\screens\zm_orderitem`

Before Adlib 5.0.3., the zoom screen you set here, was also used to display details of a linked record from the same primary data source (internally linked records), like for the hierarchical relations in the Thesaurus; if you open the detailed presentation of a term, then any broader, narrower and preferred terms and the like are displayed underlined, and if you clicked one of these, a zoom screen opened. From 5.0.3 however, when you click an internal link, you open the linked record in full detailed presentation in the current Adlib window, without making use of any zoom screen that may or may not has been set (the zoom screen is ignored). This way the user can browse between the detailed displays of all internally linked records that he or she comes across.

Zoom/edit screen

Select the screen in which Adlib allows details of a linked record from another data source to be changed when the user enters edit mode in a zoom screen (see the *Zoom screen* option above). This may be the same screen as the *Zoom screen*, and in existing Adlib applications it often is.

There is no default edit screen, so you cannot edit the data from a zoom screen without specifying an edit screen. You can also specify an edit screen for a linked field in the database setup, but a zoom/edit screen specified for a screen field has priority over any edit screen specified in the database setup. In most cases, edit screens are associated with linked fields in the database setup, because the data dictionary field is only defined once, while screen fields may appear on several screens and in different applications, but usually need the same edit screen anyway.

A relative path to a screen must be relative to the application folder, so for instance: `..\screens\zm_orderitem`

2.4.2.3 Field conditions

After you have opened a screen in the *Screen editor*, right-clicked a screen field (a.k.a. entry field) on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Field conditions* tab in that window, you can set conditions under which the current screen field must be hidden from view or must be set to non-editable.

Note that it is allowed to stack screen fields and their labels on top of each other, making it possible to show only one of those field/label combinations depending on conditions that you have set for those screen fields.

[Click here](#) for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

The Copy button

If the *Copy* button is active, it means there's a hidden field condition in the pre-6.6.0 format present. Click the *Copy* button to convert it to the 6.6.0 (and higher) format and make it visible. A hidden condition doesn't work. If the *Copy* button is greyed out, it just means there never was any condition for this field.

Suppress & Read-only conditions

From Adlib 6.0, screen fields may be conditionally hidden or set to read-only. This means that a screen field may be visible or not, or may be editable or not, dependent on the value(s) in certain other fields. This functionality is meant to be able to create more surveyable screens by only showing fields if an associated key field is filled in by the user. When a field is not visible during editing of a record as a consequence of a condition which you set here, then no strange empty spot will be present in the relevant screen, because all fields and labels underneath it will have moved one line up. And as soon as the field does become visible, the other fields will move back down again.

Suppressing fields this way is not a safe way to protect confidential information, because there would still be other ways to extract the information you want to protect. Confidential information is better protected with the proper access rights.

With the upper box you specify the conditions under which the current screen field must become/remain invisible, while in the lower box you determine when the current screen field should be read-only. Enter a single expression or a plural, Boolean expression as the

condition in either or both boxes. Everytime you open a record for display or editing, and everytime you leave a field in edit mode, all relevant expressions will be evaluated. When this screen is associated with the current data source and if the field condition is true, the screen field will be hidden or set to read-only in the detailed display of a record, while if the result if false, the screen field will be shown and/or be editable. See the general topic Conditional screens and fields for a full description of this functionality and its possibilities.

The basic form of a single expression is:

```
<field or constant> <operator> <value>
```

Example of a suppress condition:

```
S6 = English && S7 = GBP
```

Result: the current screen field will be hidden if the tag `S6` contains the value `English` and the tag `S7` contains the value `GBP`.

2.4.3 Labels

After you have opened a screen in the *Screen editor*, right-clicked a label on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Label* tab in that window, you only determine a few visual properties of this particular label.

To display a fixed text on a screen, you must place a separate label on the screen. Labels that serve to visually "name" an entry field are automatically placed on the screen when you add a new field. The difference between the two labels is that the second type is connected to a screen field (which is relevant for inserting group occurrences when editing a record), and that the first type is not connected to an entry field.

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage labels in the *Screen editor*.

On the current tab you'll find the following settings:

Captions

You must provide a caption for this label in one or as many languages as you wish to make available to users of this application. Make sure that all translations fit the length of the label on screen, so that the user won't see a half caption.

For a label connected to an entry field, leave the caption empty if for the associated data dictionary field screen texts have been specified

and you wish to use those texts here.

Justification

The labels in front of every field on an Adlib screen, are left-justified by default: this means that the text in each label always starts at the left side of the label. From Adlib Designer 6.5.38.1 (for Adlib 6.5.2 and higher), you can set a different justification per label here. Choose one of the following settings:

- *None* – uses the default setting for justification, which means: left justification.
- *Left* – justifies the text to the left side of the label.
- *Center* - justifies the text to the middle of the label.
- *Right* - justifies the text to the right side of the label.

By the way, you'll get the best layout if all fields on a screen are justified the same way.

Foreground & Background colour

Set a foreground colour and/or background colour for the current label only, if you want to set it apart from most other labels on the screen. This way you can stress the importance of certain fields, for instance. Colours that you've set for individual labels and fields, will normally not be overwritten by colours that you set on any other level.

Click the coloured box that you want to set differently, to open a standard Windows colour picker from which you can choose a new colour. To the right of the coloured boxes, you'll see an example of how the two colours plus the selected font look together.

If you would like to assign colours to all (or most) labels on this screen, or even to all screens at once, you should not use this option; click here for more information about colouring your application efficiently.

Font type

Per screen field label or box title – screen fields are commonly grouped visually in a box – you can set a font type. Click the label example (next to the foreground and background colour boxes) to open a standard Windows font type dialog window. The font type and other layout characteristics you choose here, will only be applied to the current label. You could use this functionality to clearly indicate the mandatory fields in your application, for example. Fonts that you

set this way have priority over the general font which you set via the *Change font* button in Adlib itself.

2.4.4 Boxes

2.4.4.1 Box

After you have opened a screen in the *Screen editor*, right-clicked a box on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Box* tab in that window, you specify translations of captions of this particular box, and the font of those captions. You cannot set foreground or background colours though: boxes are always transparent.

A box is for visually grouping screen elements. You can place these boxes on all types of screens. Screen elements that are placed inside a box become part of it but are not automatically assigned a group number so the frame is for cosmetic purposes only.

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage boxes in the *Screen editor*.

On the current tab you'll find the following settings:

Captions

You may provide a caption of the frame in one or as many languages as you wish to make available to users of this application. The caption will be displayed in the screen in the top left corner of this box.

Foreground & Background colour

<functionality not implemented yet>

Font type

Per screen field label or box title – screen fields are commonly grouped visually in a box – you can set a font type. Click the label example (next to the foreground and background colour boxes) to open a standard Windows font type dialog window. The font type and other layout characteristics you choose here, will only be applied to the current label. You could use this functionality to clearly indicate the mandatory fields in your application, for example. Fonts that you set this way have priority over the general font which you set via the *Change font* button in Adlib itself.

2.4.4.2 Advanced

After you have opened a screen in the *Screen editor*, right-clicked a box on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens.

A box is for visually grouping screen elements. You can place these boxes on all types of screens. Screen elements that are placed inside a box become part of it but are not automatically assigned a group number so the frame is for cosmetic purposes only.

Click here for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

Field group

In Adlib applications for DOS the group number of the box had to be the same as that for the group of (repeated) screen fields therein, to enable the box to extend when occurrences were added.

For Adlib Windows applications the group number for a box is obsolete, because the frame will extend anyway. Just leave this property at 0.

2.4.5 Images

After you have opened a screen in the *Screen editor*, right-clicked an image box on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Image* tab in that window, you must refer to the tag that holds the paths to image files and you may provide format strings for thumbnail captions.

The image tag should be defined in the data dictionary, and be of data type *Image*. And it must of course be present in the database(s) for which the currently specified screen can be opened. Typically, you also place an entry field for this tag on the screen, so that the user can either type the path to an image file to be linked to the current record, or retrieve that path by looking up an already defined reproduction record in a linked *Visual documentation* database.

In general, you must place an image element on a screen, if you want image files linked to a record to be displayed on this screen. If this screen is a detail screen, the image screen element produces the *Image viewer* window in which all linked images can be viewed. And if this screen is a brief screen, the image box produces a thumbnail image of the first linked image to each record.

The location and size you give to the image box, determine the initial location and size of the *Image viewer* if this a detail screen. (But the

user may resize and move the *Image viewer*, and can also decide to let it appear on every detail screen, even if you placed the image box on only one detail screen during design.)

Click here for information on how to edit screen object properties in general. And click here to read about how to manage image boxes in the *Screen editor*.

On the current tab you'll find the following settings:

Tag

Fill in the tag for the image reference field that holds the path(s) to one or more linked image files; in Museum applications this is usually the *Identifier (URL)* field tag *B1*. Note that tags are case sensitive.

Format strings

On the *Thumbnails* and *Filmstrip* tabs of a brief display with images (like in Museum applications), below the thumbnails by default you'll find the record number to which the image belongs. But often that is not very informative. That is why from 6.1.0 the datum from one or more other fields can be displayed, if necessary with a fixed text. With this functionality, the title from the concerning record can be shown, or the object number, for instance. In Adlib Designer 6.1.1852 and higher you can set this per brief display screen.



Title: Mary Magdalene at the Door of Simon the...



Title: Au Café

So if the currently edited screen is meant for brief display, and you want to specify which text must be displayed beneath thumbnail images on the *Thumbnails* and *Filmstrip* tabs, then with this option you can set a format string per language. This format string must be constructed as follows:

```
<fixed text>%<tag or field name>%<tag or field name>%  
etc...<fixed text>
```

The section between percent characters is the data part: enter the field name (in the proper language or in the standard language, English) or the tag of which you want to display the content. By default the first occurrence will be used. Optionally, you may provide a fixed text, before and/or after the data part, for instance to explain the datum. Examples of filled in format strings are:

```
Title: %TI%  
  
%object_number% (obj.no)  
  
%OB% %IN%
```

The data as well as the text part is optional. If you don't provide a format string, or when the specified format string is empty for a certain image, then by default the record number is displayed. When the string to be displayed is longer than two lines, the second line will be cut off with suspension points (...).

2.4.6 Menu options

After you have opened a *Brief display* screen in the *Screen editor*, you'll find a menu option (checkbox) on it. It has no properties which can be changed manually, so you can't open a *Properties* window for it. By default, there is one property attached to it, namely the reserved tag *1, which is used to mark records with.

A menu option cannot be used in detail screens. If you need a checkbox on a detail screen (if a field value can only be on or off, or be yes or no), you must use a normal screen field associated with a data dictionary tag of the *Logical* type.

[Click here](#) to read about how to manage menu options in the *Screen editor*.

2.4.7 System fields

After you have opened a screen in the *Screen editor*, right-clicked a system field on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *System field* tab in that window, you specify all properties for this control. In Adlib DOS applications, system fields were used by Adlib to display a temporarily filled reserved tag, with for instance the current dataset name and the selected access point, a counter number, a message or instruction, or in a brief display screen a sequence of various database fields.

In Adlib Windows applications system fields are still only used (and necessary) on brief display screens. All fields and labels on such a screen, or all *List fields*/parameters in the screen properties, will be concatenated in the system field.

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage system fields in the *Screen editor*.

On the current tab you'll find the following settings:

Tag

In a typical brief display screen, there is one system field with the reserved tag *A. This allows for one line of text per record. The fields you concatenate to make up this text, must therefore fit in the system field (the part that doesn't fit will not be visible). So make sure the system field is long enough on screen, to contain all text. If the concatenated text will probably be longer than one line, you may add following system fields beneath each other to allow for extra line space per record. These extra system fields must then be assigned the reserved tags *B, *C, *D and *E. So you can reserve maximally five lines of text per listed record.

Note that tags are case sensitive.

Justification

Choose *Left*, *Right*, or *Center* to specify how data in this field should be aligned. The default is *Left*.

Foreground & Background colour

You may set a foreground colour and/or background colour for the current system field only. Of course this colour will be applied to all occurrences of this system field that appear in a brief display in a running application. Colours that you've set for individual system fields, will normally not be overwritten by colours that you set on any other level.

Click the coloured box that you want to set differently, to open a standard Windows colour picker from which you can choose a new colour. To the right of the coloured boxes, you'll see an example of how the two colours look together.

If you would like to assign colours to all screens at once, you should not use this option; [click here](#) for more information about colouring your application efficiently.

2.4.8 Text windows

After you have opened a screen in the *Screen editor*, right-clicked a text window on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Text window* tab in that window, you specify all properties for this control.

In Adlib DOS applications, a text window could be used to display different fields conveniently together in a small window. This implementation used subsequent parameter fields that each held a database tag and separator text, the latter to be displayed in front of the value from the tag, to label the field.

In Adlib Windows applications it is still meant to place a small window on screens in the detailed presentation of a record to display different fields next to and beneath each other, but now with the purpose to create a sort of print preview that is continuously present, and is updated each time you leave a field. (Strictly speaking this is not really a print preview, because the image is not entirely the same as the printout, but it serves as a quick and good impression anyway.)

Since only a small window is handy, it should also be a small (print) preview, in which you display data for a label, for instance. Suppose you want to see a label preview of name and address data of each contact, on the *Address details* screen (or similar) for the *Persons and institutions* file in your application. So that during entry of that data you already know what the label with that data will look like, before you might print it later on.

The screenshot shows a software window with a tabbed interface. The active tab is 'Address details'. The main area contains a form for 'Smith, P.' with the following fields:

Address details	
Address type	
Address	154 Pine Lane
Postal code	94044
Place	Black River, CA
Country	USA

To the right of the form is a preview window titled 'Address label preview' which displays the following text:

```
Smith, P.  
154 Pine Lane  
Black River, CA 94044  
USA
```

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage text windows in the *Screen editor*.

On the current tab you'll find the following settings:

Tag

Provide the tag in which the text is stored that should be copied to the text window. This is just a database field, probably one that you put together (fill in) via an *adapl*, with data from several other fields, including line breaks.

Note that this adapt must be executed regularly during editing of a record. Therefore, set the adapt as a *Field based procedure* for the concerning database.

Captions

Enter the caption of the text window for the different languages. This caption will be shown in the title bar of the text window.

2.4.9 Web browser controls

After you have opened a screen in the *Screen editor*, right-clicked a web browser control on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Web browser control* tab in that window, you only specify the XSLT stylesheet to be used by this control. The size and position of the control cannot be adjusted by the user, so place the control appropriately and make it large enough to display the HTML page properly.

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage labels in the *Screen editor*.

On the current tab you'll find the following setting:

Stylesheet

Set an XSLT stylesheet which transforms Adlib grouped XML (of records from the database with which the current screen is associated) into HTML.

See the [Using a web browser control](#) topic for more information.

2.4.10 HTML fields

2.4.10.1 Data

After you have opened a screen in the *Screen editor*, right-clicked (the border of) an HTML field on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Data* tab in that window, you specify the data dictionary field tag, its repeatability on screen and the read/write access permission to be used by this control.

[Click here](#) for information on how to edit screen object properties in general. And [click here](#) to read about how to manage labels in the *Screen editor*.

On the current tab you'll find the following setting:

Tag

Fill in the existing data dictionary tag for this screen field. The data type of that field has to be *HTML*. Note that tags are case sensitive.

Repeatability

Choose whether this field may be *Repeated*, *Not Repeatable*, or *Repeatable*, *values must be unique*. Repeated screen fields can have more than one occurrence, whilst *Not Repeatable* can only have one. With *Repeated*, *value must be unique* every occurrence must be different from the others for the same field.

Access

When the screen itself is editable, you can still specify for individual entry fields that they are not editable, by choosing *Read only* for this property. This will apply to all users. Leave the setting to *Read and write* if users with the proper access rights must be able to enter data into the field.

2.4.10.2 Field conditions

After you have opened a screen in the *Screen editor*, right-clicked a screen field (a.k.a. entry field) on it and chose *Properties* in the pop-up menu, the *Adlib object properties* window opens. On the *Field conditions* tab in that window, you can set conditions under which the current screen field must be hidden from view or must be set to non-editable.

Note that it is allowed to stack screen fields and their labels on top of each other, making it possible to show only one of those field/label combinations depending on conditions that you have set for those screen fields.

Click here for information on how to edit screen object properties in general.

On the current tab you'll find the following settings:

The Copy button

If the *Copy* button is active, it means there's a hidden field condition in the pre-6.6.0 format present. Click the *Copy* button to convert it to the 6.6.0 (and higher) format and make it visible. A hidden condition

doesn't work. If the *Copy* button is greyed out, it just means there never was any condition for this field.

Suppress & Read-only conditions

From Adlib 6.0, screen fields may be conditionally hidden or set to read-only. This means that a screen field may be visible or not, or may be editable or not, dependent on the value(s) in certain other fields. This functionality is meant to be able to create more surveyable screens by only showing fields if an associated key field is filled in by the user. When a field is not visible during editing of a record as a consequence of a condition which you set here, then no strange empty spot will be present in the relevant screen, because all fields and labels underneath it will have moved one line up. And as soon as the field does become visible, the other fields will move back down again.

Suppressing fields this way is not a safe way to protect confidential information, because there would still be other ways to extract the information you want to protect. Confidential information is better protected with the proper access rights.

With the upper box you specify the conditions under which the current screen field must become/remain invisible, while in the lower box you determine when the current screen field should be read-only. Enter a single expression or a plural, Boolean expression as the condition in either or both boxes. Everytime you open a record for display or editing, and everytime you leave a field in edit mode, all relevant expressions will be evaluated. When this screen is associated with the current data source and if the field condition is true, the screen field will be hidden or set to read-only in the detailed display of a record, while if the result is false, the screen field will be shown and/or be editable. See the general topic Conditional screens and fields for a full description of this functionality and its possibilities.

The basic form of a single expression is:

```
<field or constant> <operator> <value>
```

Example of a suppress condition:

```
S6 = English && S7 = GBP
```

Result: the current screen field will be hidden if the tag `S6` contains the value `English` and the tag `S7` contains the value `GBP`.

2.5 Adlib screens, tags and parameters

§

2.5.1 Altering screens for DOS applications

In Adlib applications for DOS (so called character set mode applications) a number of screens are used, that today, in Adlib applications for Windows have become obsolete. You might still find them in your Adlib folders or in your application settings, but their functionality has now been integrated in the executable itself or somewhere in the application setup; so altering these screens has no effect.

When editing an Adlib DOS application however, it's necessary to know about these screens, and their functionality.

Unlike screens for Windows, for DOS you can edit not only edit brief display screens and tabs in the detailed display, but also the welcome screen, selection screens, and search screens. Modification of the following screens in particular should be handled with extreme care:

<i>setup1.fmt</i>	Selection screen for ADSETUP and DBSETUP: not used in Adlib Designer.
<i>setup2.fmt</i>	Setup screen for ADSETUP and DBSETUP: not used in Adlib Designer.
<i>setup4.fmt</i>	Setup screen for ACSETUP: not used in Adlib Designer.
<i>link.fmt</i>	Screen used to display available keys to a linked database.
<i>help.fmt</i>	Screen for displaying help texts.
<i>control.fmt</i>	The function key layout screen.
<i>controlm.fmt</i>	The shifted function key layout screen.
<i>print.fmt</i>	Selection screen for the output facilities.

<i>point.fmt</i>	Selection of pointer files.
------------------	-----------------------------

These screens were supplied with every Adlib DOS application and cannot be used for other purposes.

Some screens were supplied with Adlib as examples; it is not compulsory to use these, but their names are reserved:

<i>welcome.fmt</i>	Default welcome screen.
<i>dbselect.fmt</i>	Default dataset menu.
<i>acselect.fmt</i>	Default access points menu.
<i>search.fmt</i>	Default search screen.
<i>brief.fmt</i>	Default brief display screen.

There are also a number of module-specific screens used only in individual Adlib modules (loans, acquisitions, serials). You are, of course, free to modify these, but you should take care not to alter the operation of the modules when doing so. The names of these screens are not fixed, so it is advisable to take special care when altering screens here.

See also

Reserved tags on screens

Tags per screen type

Parameters per screen

2.5.2 Reserved tags on screens

Reserved tags make it possible to display system information on the screen in the same way as database information. The second table on this page lists the reserved tags and the screen types on which they can be used; the used abbreviations for the screen types are listed in the first table.

The following abbreviations are used to indicate the screen types in the reserved tags table:

Abbrev.	Default name	Description
----------------	---------------------	--------------------

acs	<i>ACSELECT</i>	Access point selection screen.
bri	<i>BRIEF</i>	Brief display of selected records.
con	<i>CONTROL</i>	Screen for displaying function keys.
dbs	<i>DBSELECT</i>	Datasets selection screen.
det	<i>DETAIL</i>	Detailed display of a record.
edi	N/A	Detailed display in edit mode.
hel	<i>HELP</i>	Help window.
lin	<i>LINK</i>	For displaying key values from a linked file.
pri	<i>PRINT</i>	Selection screen for print options.
poi	<i>POINT</i>	Pointer files selection screen.
sea	<i>SEARCH</i>	Search screen, index list display.
su1	<i>SETUP1</i>	Menu screen for setup programs.
su2	<i>SETUP2</i>	Definition screen for parameters in the setup programs.
wel	<i>WELCOME</i>	Application welcome screen.

Most of the screen types listed above are no longer used in Adlib applications for Windows. See the Help topic Altering screens for Adlib DOS applications, for more information.

Reserved tags on Adlib screens

Unless indicated otherwise, system fields on a screen contain reserved tags.

Most reserved tags can only be used on screens in Adlib applications for DOS; reserved tags that can be used on screens in Adlib applications for Windows too, are marked in light green. (In Adlib applications for Windows, most functionality that under DOS was implemented through tags or parameters on screens, is now an integral part of the software, or can be set through properties of Adlib objects.)

The number between brackets indicates the maximum number of

occurrences that the tag can have; *(n)* means that there is no maximum. The number after the hash sign (#) refers to the serial number in the text files for the Adlib software.

Tag	Content	Screen(s)
&1(2)	Search key/file name (input)	sea, pri, det
*0(1)	Program title (#1)	All screens
*1(<i>n</i>)	Checkbox (menu option)	bri, acs, dbs, sea, hel, lin, poi
*2(1)	Dataset title	acs, sea, bri, det
*3(1)	Access point	bri, det
*3(2)	2 prompt lines	sea
*4(<i>n</i>)	System message	con, hel
*5(8)	Function keys: labels	con
*6(8)	Function keys: functions	con
*7	<i>(not used)</i>	
*8(1)	Match number	det
*9(1)	a. Screen number (x of y)	det, hel
	b. Number of matches	sea, bri
*A(1)	Record line 1: the content of all database tags on a brief display is concatenated and by default displayed on one line. If the total content is longer than one line on the brief display screen - the length of this line is only determined by the width of your Adlib application window, not by the length of the system field - then the content is cut off at the end of the line if you only provide the system field *A.	bri
*B(1)	Record line 2: if it is likely that the concatenated content of database tags	bri

	for the brief display is longer than one line, then insert the system field *B below system field *A, to allow said content to word-wrap to a second line. Note that each record on the brief display takes up two lines now, even if the content fits on one line.	
*C(1)	Record line 3: if *A and *B still don't offer enough line space to display the contents of all tags on the brief display, then add further system fields *C and possibly *D and *E to create more room for the database tags.	bri
*D(1)	Record line 4	bri
*E(1)	Record line 5	bri
*F(1)	File name prompt	pri, det
*G(1)	Edit TO (#127)	bri
*H(1)	Date/time changed	det
*I(1)	Marked (#138)	det
*J(1)	Search time status	sea, bri
*K(1)	Access key	bri
*L(1)	Marking status	pri, bri, det
*M(1)	Last used priref	sea
*N(1)	Selected option number	sea, bri, lin, poi
*O(1)	Insert ON/OFF	det, pri, sea
*S(1)	Field status	con (only for det)
*T(1)	Trace On	select (search language)

See also

Tags per screen type

2.5.3 Tags per screen type

Depending on the purpose of an Adlib screen, certain reserved tags are either allowed or mandatory. Below is a listing of these tags for each screen. The screen names used are the default names for that screen type. Most of these screens and reserved tags can only be used in Adlib applications for DOS; only the tags and parameters marked in light green can also be used in Adlib applications for Windows. See the Help topics *Altering screens for Adlib DOS applications* and *Reserved tags on screens*, for more information. (In Adlib applications for Windows, most functionality that under DOS was implemented through tags or parameters on screens, is now an integral part of the software, or can be set through properties of Adlib objects.)

The number between brackets behind a tag is the maximum number of occurrences of that tag; (*n*) means that there is no maximum.

Screens: WELCOME

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
	parameter	see Parameters per screen
*0	system field	program title from text file

Screens: DBSELECT

Tag	Object type	Comments
	label	fixed text
	box	cosmetic
*0	system field	program title from text file
*1(n)	menu option	dataset titles

Screens: ACSELECT

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*0(1)	system field	program title from text file
*1(n)	menu option	access point titles
*2(1)	system field	dataset title
*E(1)	system field	With old executables this reserved tag displayed text to indicate a combine search status, but now this is displayed by default in the tab header. Therefore *E has no longer a function here.

Screens: SEARCH

Tag	Object type	Comments
	label	fixed text
	box	cosmetic
&1(2)	field	1 or 2 fields for entering a search key. The occurrence number can be either 1 or 2. You only need to define one input field for entering search keys in ordinary text fields (term index) and free text fields (word index). For numeric searches or searches on date, you will need two input fields.
*0(1)	system field	program title from text file
*1(n)	menu option	serial numbers from index values
*2(1)	system field	dataset title
*3(2)	system field	prompt text for the input fields

*9(1)	system field	match counter
*E(1)	system field	With old executables this reserved tag displayed text to indicate a combine search status, but now this is displayed by default in the tab header. Therefore *E has no longer a function here.
*J(1)	system field	show search time
*M(1)	system field	last used record number
*N(1)	system field	option number entered by user
*O(1)	system field	insert on/off text

Screens: BRIEF

Tag	Object type	Comments
	label	fixed text
	box	cosmetic
*0(1)	system field	program title
*1(n)	menu option	checkboxes and record numbers of records in the selection
*2(1)	system field	dataset title
*3(1)	system field	access point title
*9(1)	system field	match counter
*A(1)	system field	first line for brief display of records
*B(1)	system field	second line for brief display of records
*C(1)	system field	third line for brief display of records
*D(1)	system field	fourth line for brief display of records
*E(1)	system field	fifth line for brief display of records
*J(1)	system field	show search time

*K(1)	system field	show selection criteria
*L(1)	system field	show number of marked records
*N(1)	system field	option number entered by user
**	parameter	field separator for brief display of records
XX	parameter	field and field line separator for brief display of records
EDIT	parameter	edit functions available (see Parameters per screen)

Screens: DETAILED

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*0(1)	system field	program title
*2(1)	system field	dataset title
*3(1)	system field	access point title
*8(1)	system field	show record counter
*9(1)	system field	(not in edit mode) counter: <i>Screen x of y</i>
*F(1)	system field	prompt name of file being imported
*G(1)	system field	during input: "Input/Edit ON"
*H(1)	system field	date/time updated
*I(1)	system field	marked
*L(1)	system field	...records marked
*O	system field	insert on/off text (in edit mode)

	text window	With old executable this reserved a space for "stacked" fields. Today, a text window has other functionality.
XX(n)	field	database field from the record
YY	parameter	tag with start text at beginning of entry field
DISPALL	parameter	Show all fields on this screen, even when they are empty. (In Designer this parameter is invisibly present on a screen, through the marked <i>Show empty fields</i> screen property.)
EDIT	parameter	If this parameter is on a screen, the user can edit the data on it, if no other restricting access rights apply of course. (In Designer this parameter is invisibly present on a screen, through the unmarked <i>Read-only</i> screen property.)
SCROLL	parameter	scroll function on

Screens: PRINT

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*0(1)	system field	program title
*1(n)	menu option	list with format or file names
*F(1)	system field	enter file name
*L(1)	system field	... records marked for output
*O	system field	insert on/off text
&1(1)	field	enter file name

Screens: CONTROL & CONTROLM

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*4(1)	system field	instruction line
*5(8)	system field	function keys: name
*6(8)	system field	function keys: function
*S	system field	field status

Screens: HELP

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*1(n)	system field	help text line from .HLP file
*4(1)	system field	instruction line
*9(1)	system field	counter: <i>page .../...</i>

Screens: LINK

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*1(n)	system field	lines with retrieved index values
*N(1)	system field	option number entered by user

Screens: point

Tag	Object type	Comments
	label	fixed information
	box	cosmetic
*0(1)	system field	program title
*1(n)	menu option	list of pointer files
*2(1)	system field	dataset title

2.5.4 Parameters per screen

Most of the screens and parameters listed below, can only be used in Adlib applications for DOS; only the parameters marked in light green can also be used in Adlib applications for Windows. See the Help topics *Altering screens for Adlib DOS applications* and *Reserved tags on screens*, for more information.

In Adlib applications for Windows, most functionality that under DOS was implemented through tags or parameters on screens, is now an integral part of the software, or can be set through properties of Adlib objects. More specifically, the `EDIT` and `DISPAL` parameters are now visible as screen properties *Read-only* and *Show empty fields* on the *Screen* tab, while all other parameters appear in or can be added to the parameters/tags list on the *List fields* tab of the screen properties.)

Parameters in the welcome screen.

A number of parameter objects can be added to the welcome screen to customize Adlib for DOS in a number of ways.

Parameter	Default	Function
BRDPALW	<i>Off</i>	Brief display always: show brief display screen even if only one record is found.

BUTTON	<i>Off</i>	With this parameter and the value <i>big</i> , all buttons are displayed double size.
COMBDISP	<i>Off</i>	Combine display: show index list during combining of search.
DBSORT	<i>Off</i>	Database sort: sort the datasets in the data source list in step 1 of the <i>Search wizard</i> .
DTDPONE	<i>Off</i>	Detailed display one: start each record with the first detailed display screen.
EMPKEYOK	<i>Off</i>	Empty key OK: empty search keys are valid.
EXPHELP	<i>Off</i>	Expert help (F1) is displayed by default.
FTRNCON	<i>Off</i>	Free text truncation on: automatically truncate free text.
HELP	<i>Off</i>	Installs F1 = Help in the welcome screen.
LANGUAGE	<i>0</i>	The number of the language used at start-up.
MAXKEYS	<i>5000</i>	Limit for the number of search keys displayed.
MAXRECS	<i>5000</i>	Limit for the number of records displayed. Adlib reserves memory space that corresponds to the number entered here. If you set this value too high, Adlib may give memory errors (error code 2).
MILSTONE	<i>100</i>	Milestone: changes the increment of the record and index counters.
NOALLKEY	<i>Off</i>	The <i>All keys</i> button is not available.

NOCOMBIN	<i>Off</i>	No combine: prevents selections from being combined.
NOHLPRET	<i>Off</i>	No help and return: suppresses Help and Return options.
NOMENU	<i>Off</i>	No menu bar on screen.
NOPRINT	<i>Off</i>	Disables print functions. As value, provide the place where you cannot print to, in the separator texts property. Use P for printer, F for file, and S for screen (use capitals).
QUIT	<i>Disabled</i>	Installs FB = Stop in the welcome screen.
STRTNOCR	<i>Off</i>	Start not CR: suppresses stating with the Enter key and replaces the text <i>Press Enter to continue</i> with <i>Press any key to continue</i> .
TIMEOUT	5	Changes the allowed period of keyboard inactivity (in minutes). A time-out of 0 means that there is no time-out.

Parameters in the search screen

Parameter	Function
QUICK1	Value to search for when the corresponding access point is called (if only one value can be entered in the search box) or value to start searching from (if you can enter a <i>from</i> value and a <i>to</i> value in the search box).
QUICK2	Value to search up to when the corresponding access point is called (can only be used if you can enter a <i>from</i> value and a <i>to</i> value in the search box).

Parameters in the brief display

Parameter	Function
EDIT	<p>Enables editing and deleting of selected records from this screen (if not disabled by other access rights). By including the <code>EDIT</code> parameter on Adlib DOS screens or unmarking the <i>Read-only</i> option in Adlib Windows screen properties, you allow the data to be edited in a given screen. This includes access to the functions F11 (<i>Edit</i>), Delete, and Shift-Delete (Delete marked records).</p> <p>If the Adlib user does not have write access to the dataset/database, the <code>EDIT</code> parameter will not override this. The user will remain unable to edit or delete data.</p>

Parameters in the detailed display screen

The following parameters can be used in the detailed display screens:

Parameter	Function
DISPALL	<p>Displays all fields, even when empty of data. In Designer, the <code>DISPALL</code> parameter on a screen is visible as the marked <i>Show empty fields</i> screen property,</p>
SCROLL	<p>Enables scrolling within repeated fields.</p>
EDIT	<p>Enables use of record edit mode from this screen. In Designer, the <code>EDIT</code> parameter on a screen is visible as the marked <i>Read-only</i> screen property.</p> <p>If an Adlib user does not have write access to the dataset/database, the <code>EDIT</code> parameter will not have any effect, i.e. editing will remain impossible for that user.</p>
AUTOEDIT	<p>Enables edit mode without first having to press F11. The screen must also contain the <code>EDIT</code> parameter.</p>

SCRMERGE	<p>Makes it possible to display fields from two screens on one screen.</p> <p>Use the <code>SCRMERGE</code> parameter to display the fields of two separate screens as one screen. This is achieved by linking the second screen to the first screen and fields not contained on the first screen are added to the first screen from the second screen. A possible application of this would be to display all different fields on the first screen and all identical fields on the second screen of the composite screen (e.g. a frame, transaction date or account flag). This would mean that if any of the fixed data changed, it would only have to be changed in one screen instead of in all the screens where it occurs.</p> <p>The link is thus provided by the screen merge parameter <code>SCRMERGE</code>. In the first screen, you should set this parameter in the <i>Tag</i> list and for its separator text you should enter the name of the second screen (i.e. the screen with which you want to merge the first screen).</p> <p>Only fields on the second screen that are not contained on the first screen will be displayed. If a field is present on both screens, only the field from the first screen is used.</p> <p>Note that this parameter needs a text window on the screen, to work.</p>
----------	--

2.5.5 Designing different screen types

Adlib screens must satisfy certain requirements, to function properly as a detailed display screen, a brief display screen, a zoom/edit screen, a search screen (*Query by form*), or a link screen. When studying your existing application and screens, there are some points that may need clarification. In this Help topic, some of the loose ends will be tied together.

In Adlib for Windows there are no default screens, except an implicit default for link screens (see below).

Although not relevant for the functioning of a screen, its name is usually indicative of its use; [click here](#) for an introduction to screens.

Detailed display screens and zoom/edit screens

In principle, zoom and edit screens are not different from detailed display screens, but for a linked record in another data source you can only display one zoom and edit screen, while for records in the current data source you usually display a number of detail screens that together hold all data from a record.

Zoom and edit screens can be associated with either data dictionary fields or screen entry fields, through their properties. Detail screens (and brief display screens too) must be linked to a data source in an application definition by listing references to the desired screens underneath a data source *Screens* node or an individual method node.

A detail, zoom or edit screen typically contains several boxes directly below each other, to visually group entry fields and labels. The order in which you place new objects on a screen is not relevant; if objects overlap, for instance if a box hides fields underneath it, you can always change the stacking order and/or move objects away.

In Adlib Windows applications, detail screens edited in Designer contain no (visible) parameters like they did in DOS. The only two relevant parameters that may be present on the screen are *DISPALL* and *EDIT*, but you'll only find these in a different form as the screen property options *Show empty fields* and *Read-only*. A typical detail screen for library is *booka.fmt (Title, author, imprint, collation)*, or *objid-in.fmt (Identification)* for museum - both to be found in the *\screens* folder or in their respective application folders.

The screenshot shows a form titled "Title, author, imprint, collation". The form is organized into several sections:

- Title and author**:
 - Title: Input field with label "lw" and "ti" below it.
 - Statement of responsib.: Input field with label "AV".
 - Author: Input field with label "au".
 - Illustrator: Input field with label "IL".
 - Corporate author: Input field with label "ca".
 - Conference: Input field with label "Co".
- Role**: Three input fields with labels "ro", "ri", and "cr" respectively.
- Edition**:
 - Edition: Input field with label "ed".
- Imprint**:
 - ISBN: Input field with label "ib".
 - Place: Input field with label "pl".
 - Publisher: Input field with label "ui".
 - Year: Input field with label "ju".

For some detail screens (depending on the application and data source), a *Before-screen adapl* has been set in the screen properties, for instance *dispcat* for the library catalogue screens. This particular

adapt empties some fields that must contain unique values after the user has copied a record, and it formats and concatenates data from the *Title* and *Author* fields, and puts it into another tag, in this case *BR*. On top of all detail screens (except in *booka.fmt*) this short reminder of what record the user is viewing or editing, is displayed in a read-only entry field associated with that database tag *BR*.

Search (Query by form) screens

A search screen is built up like a detail screen, but the entry fields on a search screen must be associated to data dictionary fields in which the user can search, so indexes for all these fields are necessary.

In a running application, there are two situations where a *Query by form* can or must be used to search:

- After a user has chosen a data source, there is of course the rest of the *Search wizard*, but you can also specify that the user is allowed to use a general *Query by form* to search the current data source, and/or to use the *Expert search system*. These options must be specified explicitly by listing a method of each of these types, for the data source, in the same list as the access point methods. In the properties of the *Query by form* method you specify the *Menu texts* that will appear in the *Start > Query by form* submenu of a running application. In the screen list underneath the *Query by form* method node in the *Application browser*, you typically define only one reference to a qbf screen. The results of a qbf search by the user will be displayed in the brief display screen specified for the current data source.
- If a user wants to derive a record from a friendly database, that record has to be searched for first. The normal way of doing so is through a *Query by form* search screen. This screen must then be referenced in the *Search screen* property of the concerning friendly database. (Friendly databases are specified underneath the data source node in an application definition, from where deriving will start.)
The results of a qbf search by the user will be displayed in the *Linked record search screen* window (see the *Adlib User guide*), because it concerns results from a linked database; you need to specify a link screen (see below) for the *Search result screen* property of the friendly database, if you don't want the *Linked record search screen* window to just display a list of record numbers that result from the user's search.

Brief display screens

A brief display screen is used for presenting a selection of records, the search result if more than one record has been found. Brief

display screens must be linked to a data source in an application definition by listing a reference to the desired screen underneath a data source *Screens* node or an individual method node.

There are three ways of constructing a brief display screen, but the first part is the same for all three methods:

On the left side of the screen, place one menu option (checkbox) and assign it the reserved tag *1: the software will use the binary value in *1 to determine whether the user marked a record or not.

Next to the menu option, on the right, place at least one system field and assign it the reserved tag *A: this reserves the space of one line to display data from each listed record. (If you want to reserve more lines, because you want to display lengthy data for instance, place one, two, three or four subsequent system fields underneath the first, and associate respectively the tags *B, *C, *D, and *E.) Then use one of the following three methods to complete the screen:

- **Only system fields** (method 1) - To specify what data and fixed text must be concatenated and displayed through *A etc., open the properties for the screen itself and on the *List fields* tab specify the desired database tags and separator texts. In an old version of the br_thes.fmt file (*Brief display thesaurus*) for instance, the following tags and texts may have been defined:

Tag	Text (English)
te	
**	(
do	/
**)

In a running application this could result in for instance:

book (documentation type)

or

book (documentation type/library object)

Text for ** separates values from database tags, while text for the tags separates any occurrences of those tags.

- **Entry fields and labels** (method 2) - To specify what data and fixed text must be concatenated and displayed through *A etc., you can also place normal entry fields and labels on this screen: the entry fields must be associated with the database tags that you want to display, the labels with the fixed texts. Currently there is no way to tell the difference between a label connected to an entry field, which holds field occurrence

separator text, and a separate label that contains field separator text (bug ref.no: 1735).

The entry fields and labels that you place on the screen can be next to each other or underneath each other. The space between is not relevant, nor is the length of the entry fields, but the order of their placement is (which is determined from left to right and top to bottom), because it determines the order in which the data and texts are concatenated.

The `br_objec.fmt` file (*Brief display objects*) for instance, looks somewhat like this:



Note that this way of constructing a brief display screen does not use the *List fields* tab in the screen properties.

- **Using an adapl** (method 3) - Using the first method to construct a brief display screen, you can also list just one tag on the *List fields* tab, and have this tag filled by a *Before screen adapl* that you set for the screen. This allows you more freedom in the concatenation, and the possibility to use the same brief screen for different data sources.

Old versions of the `br_catal.fmt` file (*Brief display* for several library data sources) for instance, have only a menu option and two system fields, `*A` and `*B`, beneath each other. In the screen properties the *Before screen adapl* `brformat` has been set, and on the *List fields* tab only the database tag `BR` is listed (without separator texts); the `adapl` formats and conditionally concatenates data from several fields together with fixed separator texts, and puts it into another tag, in this case `BR`. In newer versions of this screen, the tag `BR` is not listed on the *List fields* properties tab, yet is present in an entry field on the screen:



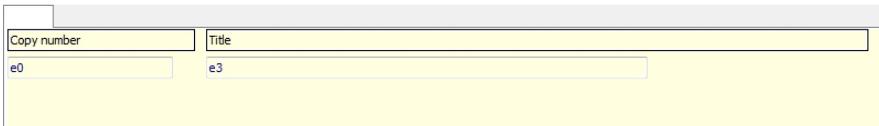
Note that you cannot use the before screen adapt to sort the records that will be listed on this screen.

Link screens

Although it is no longer recognizable as a screen, the *Find data for the field...* window (better known as the *Linked record search screen*) can use a link screen (associated with the linked data dictionary field) to determine what fields to list on the *View table* tab when the user presses **Shift+F4** or clicks the *List* button in the relevant linked field.

A link screen contains only two lines of screen elements. The top line consists of labels for the entry fields below them in the second line. Adlib converts this link screen to the *View table* tab in the *Linked record search screen*: with the labels on the link screen you specify the column headers, and with the fields the contents of the columns on that tab.

The `Ink_copy.fmt` file for instance, which is used in a library application when the user asks for a list of values in the `copy.number` field on the *Copies and shelf marks* tab. Since a list of just all copy numbers is not informative, the user gets to see the title of the document and an associated copy number, for each registered copy. So if a title has four copies, it appears in the list four times, with four different copy numbers. The layout of `Ink_copy.fmt` is as follows:



You can see two labels on top, and two normal entry fields below them (each label is connected to the entry field below it). The label texts must of course be entered in as many languages as you wish to make available to users of your application. The screen properties themselves are default, except that the *Screen descriptions* are empty and the *Show empty fields* option has been unmarked.

3 Application setup

§

3.1 Applications

When we normally speak of an Adlib application or module in our user guides or brochures, we generally mean the application that you see on the screen of your monitor when you work in Adlib. These applications include the Adlib Library, Museum, or Archive applications, or the modules Reproduction orders or Serials, for example.

These applications and modules that have the standard Adlib user interface (of which the functionality is described in the general Adlib User guide) run on *adlwin.exe* (see next paragraph) and can be created or edited in Adlib Designer. (So, this excludes our Internet Server applications, that have no custom Adlib editor.)

The Adloan Circulation application is a special Adlib workflow application that runs on *adloan.exe*. Its graphical user interface is different from the "adlwin" applications, and especially suited to process loans, returns and reservations at a library loan desk. From 6.5.0, this type of application can also be created and/or edited in Designer.

The components of an Adlib application

Adlib applications are actually a symbiosis of a number of very different files, distinguished most notably by on the one hand a files collection of information about the interface structure of your application (mainly in the *.pbk*, and *.fmt* files), and on the other hand the software files that use said information to run the application and display it on your monitor (mainly *adlwin.exe* and associated *dll's*). The software offers all the functionality, like printing, searching and editing, while the interface structure specifies which databases, screens, methods and adapls and such will be used by that functionality.

The software is regularly updated by Adlib through service-packs, which offers you new functionality in your existing applications.

You cannot edit this software in any way.

Your existing applications are never updated by Adlib (unless you commission Adlib to do so for your application), because you may have customized your application, and an update would overwrite

any of those modifications.

Application and database definitions

Although everything you edit in the tools of Designer is considered part of an Adlib application, we speak of the "application setup" when we mean to edit the application **definition**: this is the main interface file - it has the extension *.pbk* and is usually executed by *adlwin.exe* (only *adcirc.pbk* is executed by *adloan.exe*) to run the application. This file holds all settings for the interface and lots of references to other files (like databases, screen files and adapls) and objects in those files (like datasets and fields). These application definitions must be edited in the *Application browser* and can be found in folders with the, sometimes cryptic, name of the Adlib applications.

Note that in the *Application browser* you can also edit the database (structure) definitions that are stored in *.inf* files, which are always located in your *\data* folder: this is called the "database setup". Keep this distinction in mind when you edit an application, because database definitions and the databases themselves should be seen as completely separate from the application definition, and treated that way. This is because a database can be used by more than one application, and its structure should only be edited if e.g. you want to be able to save a new field in it, or need extra domains in an authority file like the thesaurus, or if you want to save a different type of value in an existing field. Most interface related stuff can be set in the application definition and in screen files and adapls, for which you never have to open the *\data* folder.

Screens and adapls

Screens and adapls are also separate files that are often used in more than one application or module. And thus, although screens are typically a part of the interface, you shouldn't edit a screen before knowing in which applications, and where, this screen is being used. (You can look for such references with the *Object searcher* in Designer.)

3.2 Data sources

A data source is a reference in the application definition to a physical database (its definition) and possibly a dataset therein. The data sources that you specify in an application, refer to the databases and/or datasets that may be accessed from the running application. If the access rights to this data source for a specific user allow it, the data source will be listed in the first step of the *Search wizard* in a running application. From this list the user selects the

database or dataset in which he or she wants to search for or edit records.

See also

Adlib file types and folders

Accessing the application setup

Managing application objects

Data source properties

Data source access rights

Screen references for data sources

3.3 Methods

In a data source in the application definition you specify so-called methods. Methods can be divided conceptually into access points and functionality switches:

- Access points are methods that refer to indexes defined in the database setup, and offer the user the possibility to search a particular index in the current database for a keyword or part thereof. Access points are listed in the step 2 of the *Search wizard* in a running application; after the user has chosen a data source in the step 1 of the *Search wizard*, the access points for it will be listed, so that the user can search for e.g. an author, an object, a record number, etc.
- Functionality switches do not appear in the access points list, but offer software functionality to the user. To include a functionality switch in the methods list for a data source, means offering that functionality to the user when he or she selects this data source in the application. The functionality you can offer this way, or leave out, consists of creating new records, deleting, advanced searching, exporting and importing, printing, deriving, or working with pointer files. If you leave out functionality, the corresponding buttons or options in the Adlib software will not be present.

Note that every method that you add to a data source can be excluded (hidden) to certain users by associating role access rights with the concerning method.

See also

Accessing the application setup

Managing application objects
Method properties
Method access rights
Screen references for methods
The Adlib User guide, for software functionality

3.4 Screens

Screens are the tab sheets or forms in Adlib applications that contain fields and labels that display information from a record, allow data input, or that you use to perform an advanced search.

Throughout an application directory in the *Application browser* in Designer you may encounter screen files and screen references. It's important to separate the two kinds:

- Screen files have the following icon:



An application folder may contain screen files that are meant for use in that application only, but the presence of those files doesn't mean that they are actually linked to the application: they are just Adlib screen files in a folder, and in newer applications you will only find those screen files in the `\screens` folder.

- References to screens have any of the following icons:



References to screen files are part of an application definition. You specify these references for data sources and/or methods in the tree view in the *Application browser*, to determine exactly which brief screens and which detail screens should be used to either display the search result list, to display data from one record in the data source, or to display a particular *Query by form* for searching.

See also

Adlib file types and folders
Accessing the application setup
Managing application objects
Screen file properties

Screen references for methods

Screen references for data sources

3.5 Output jobs

Output jobs or output formats are adapls and/or Word templates, or XSLT stylesheets that are used to print data from marked records in a specified way (selected fields in a specific layout). All the output jobs that you specify for a data source will be listed in the Adlib *File > Output formats* menu.

For an output job you can use an XSLT stylesheet, an adapl, a Word template, or a combination of adapl and Word template:

- **XSLT stylesheet** - For creating any type of report, simple or complex, one or more XSLT stylesheets may be put to use (from Adlib 6.5.0). XSLT (Extensible Stylesheet Language Transformations) is a standardized language (an XML variant) for converting an XML document to a differently structured XML document or to a document in another format, for instance an HTML or text document. During transformation, the data from the original XML document can also be processed in other ways. Moreover, XSLT has programming language characteristics like variables and functions. Adlib stores records as XML and when you execute an XSLT output job, this XML is passed on to the stylesheet which converts the XML to the desired format: this target format would need to be HTML if it concerns an output format (Adlib will actually print the generated HTML page). Until Adlib 7.1, the source Adlib XML format passed on to XSLT stylesheets was of the unstructured type by default. Since this type has its drawbacks, Adlib 7.1 (and higher) is capable of passing on Adlib XML of the grouped type as well. Which XML type must be generated by Adlib, can be set per XSLT *Output job* in the *XML type* option. Adlib XML is formatted according to the *adlibXML.xsd* (an XML Schema Definition). The most important thing you need to know about Adlib XML is that only the XML tags of the three highest levels have been defined, namely: `adlibXML` (root tag), `recordlist` (may occur only once), `record` (may occur indefinitely). Further, there is a `diagnostic` tag on the level of the `recordlist`, which contains metadata. The structure of an Adlib record itself is not defined in the schema definition because this differs per database and XML type. It's easy though, to obtain an example of Adlib XML, by exporting a few marked records to an *Unstructured* or *Grouped* XML file from your Adlib application: see the *Importing and Exporting* chapter in the Adlib User Guide for more information about exporting from within Adlib. The main

advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence. Whenever you create an XSLT stylesheet for unstructured Adlib XML, which must be able to collect field data per field group occurrence, you have no choice but to always count the "position" of every processed field occurrence because that's the only way to retrieve the other fields from the same position. In grouped Adlib XML on the other hand, there's no need for such a workaround because every field group occurrence is contained within its own field group node. Matching an XSLT template to a field group node automatically provides access to all grouped fields with the same occurrence number (in other words: at the same position).

Further note that only XSLT version 1.0 is supported, not XSLT 2.0. Of XML, both version 1.0 and 2.0 are supported, and XSLT 1.0 can be used in an XML 2.0 document if needed.

More information about using XSLT can be found in the separate *Programming XSLT stylesheets for Adlib* document.

- **Adapl** - For creating advanced reports, for which pre-processing of data is necessary, the Adlib programming language ADAPL can be used as well as XSLT. Current Adlib applications all use ADAPL programs, but if you feel that the non-standard nature of the Adlib proprietary ADAPL programming language limits the maintainability of your output formats, then you may choose to build your output formats in XSLT. However, in combination with a Word template, any complex calculations, manipulations of field contents, etc. will have to be managed by an adapl. Output adapls that do not use Word templates, will also have to manage the layout of the data to be printed, with `PRINT` and `OUTPUT` statements.
- **Word template** - Microsoft Word can be used by Adlib very conveniently for creating mailings, labels and documents in which the company style of an organisation is very important. For this purpose, default Word templates can be edited to enable printing of Adlib data with such a template. Documents that are created this way, can be saved, printed or sent through e-mail. You can set up all available Word templates as output formats if you like; that is certainly more user-friendly than letting the user search for an applicable template when printing (which is always possible). But only for label templates it is really necessary to set up the Word template as an output format. See the Adlib User

Guide for more information about creating a Word template for Adlib.

- **Adapl and Word template** - To use this combination, you can choose to set both an adapl and a Word template in the properties of an output job, in which case a Word document is filled with data after the adapl for pre-processing the record data has been carried out, or just set the adapl and no Word template, in which case the Word template must be called from within the adapl. The difference to the print result is that if you generate or print Word documents from within an adapl, you can choose if and when to call the template, while setting the template in the output job properties means it will be called automatically every time after processing a record with the adapl. In both cases, the adapl should not contain any `PRINT` or `OUTPUT` statements.

See also

Accessing the application setup

Managing application objects

Output job properties

Output job access rights

3.6 Export formats

Export formats are adapls (similar to output adapls) and/or Word templates, or XSLT stylesheets, with which you generate an exchange file from selected records, in a format that is completely up to you. Exports formats that you specify in the application setup for a data source are listed below the standard export types in the *Export wizard* in a running Adlib application. In the running application, the *Export* option in the *File* menu can be used as soon as one record (in the detailed presentation) or more records (in the brief display) have been marked. The selected records can be exported to any of the listed formats. You create an export adapl if the default formats are not what you want, for instance if you want to use different field or record separator characters in the exchange file to be created. Note that export formats must not be confused with export jobs which consist of the properties of an actual export procedure that can be run from within the *Export job editor* in Designer.

- **XSLT stylesheet** - For creating any type of exchange file, simple or complex, one or more XSLT stylesheets may be put to use (from Adlib 6.5.0). XSLT (Extensible Stylesheet Language

Transformations) is a standardized language (an XML variant) for converting an XML document to a differently structured XML document or to a document in another format, for instance an HTML or CSV document. During transformation, the data from the original XML document can also be processed in other ways. Moreover, XSLT has programming language characteristics like variables and functions.

Adlib stores records as XML and when you execute an XSLT export format, this XML is passed on to the stylesheet which converts the XML to the desired format: this target format can be any desired format (XML, HTML, CSV, Adlib tagged, plain text, etc.) if it concerns an export format. Until Adlib 7.1, the source Adlib XML format passed on to XSLT stylesheets was of the unstructured type by default. Since this type has its drawbacks, Adlib 7.1 (and higher) is capable of passing on Adlib XML of the grouped type as well. Which XML type must be generated by Adlib, can be set per XSLT *Export format* in the *XML type* option. Adlib XML is formatted according to the *adlibXML.xsd* (an XML Schema Definition). The most important thing you need to know about Adlib XML is that only the XML tags of the three highest levels have been defined, namely: *adlibXML* (root tag), *recordlist* (may occur only once), *record* (may occur indefinitely). Further, there is a *diagnostic* tag on the level of the *recordlist*, which contains metadata. The structure of an Adlib record itself is not defined in the schema definition because this differs per database and XML type. It's easy though, to obtain an example of Adlib XML, by exporting a few marked records to an *Unstructured or Grouped XML* file from your Adlib application: see the *Importing and Exporting* chapter in the Adlib User Guide for more information about exporting from within Adlib. The main advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence. Whenever you create an XSLT stylesheet for unstructured Adlib XML, which must be able to collect field data per field group occurrence, you have no choice but to always count the "position" of every processed field occurrence because that's the only way to retrieve the other fields from the same position. In grouped Adlib XML on the other hand, there's no need for such a workaround because every field group occurrence is contained within its own field group node. Matching an XSLT template to a field group node automatically provides access to all grouped fields with the same occurrence number (in other words: at the

same position).

Further note that only XSLT version 1.0 is supported, not XSLT 2.0. Of XML, both version 1.0 and 2.0 are supported, and XSLT 1.0 can be used in an XML 2.0 document if needed.

More information about using XSLT can be found in the separate *Programming XSLT stylesheets for Adlib* document.

- **Adapl** - For creating advanced exchange files, for which pre-processing of data is necessary, the Adlib programming language ADAPL can be used as well as XSLT. Current Adlib applications use ADAPL programs for many purposes, but if you feel that the non-standard nature of the Adlib proprietary ADAPL programming language limits the maintainability of an export format yet to be created, then you may choose to program it in XSLT. Export adapls will have to manage the structure and layout of the data to be exported, using `PRINT` and `OUTPUT` statements.
- **Word template** - Microsoft Word can be used by Adlib very conveniently for creating mailings, labels and documents in which the company style of an organisation is very important. For this purpose, default Word templates can be edited to enable printing of Adlib data with such a template. Documents that are created this way, can be saved, printed or sent through e-mail. Word templates are not commonly used as export formats though.

See also

Accessing the application setup

Managing application objects

Export format properties

3.7 Friendly databases

A "friendly" database is a database or dataset from which the user can "derive" records into the current data source in an Adlib application (*Record > Derive record...*): this means the user can search in a friendly database and move or copy a record into the current data source when the latter has been opened by the user.

Deriving records from other databases or datasets is very useful when cataloguing items that contain comparable data.

An example of a friendly data source is the `serials` dataset for the `book` dataset in the `document` database of a library application; `serials` will then be the dataset from which data can be derived into the `book` dataset.

A friendly database doesn't necessarily need to be located on the

same computer or even on the same local network: a remote, third-party database on the internet may be suitable to function as a friendly database to Adlib as well. From Adlib 6.5.0, access to selected external sources is offered in the form of source-specific gateways (HTTP handlers) hosted by Axiell ALM Netherlands.

See also

- Accessing the application setup
- Managing application objects
- Friendly database properties
- Friendly database access rights

3.8 Users and roles

If you want to restrict access to certain parts of an Adlib application for certain users, you must register all possible users in the concerning application definition. Since Adlib recognizes a user by his or her login name in Windows (the name used by a user to log in to a stand-alone computer or local network on which the Adlib application is used), you must register those exact same names for all users of that computer or network who's access needs to be limited. You can register everyone of course, but this is not really necessary, since non-registered users have full access to the Adlib application by default.

To register a user in an Adlib application, you create a new user in the application setup for a specific application and assign it a so-called role, which is a group name for (usually) multiple users that must have equal access rights. You can choose new role names or select existing ones.

One special role is `$REST`. This role should never be assigned to users explicitly. Its purpose is to be able to specify access rights to the current object for all users who's role has not been assigned access rights to this object and for all users without a role. So if you've defined users and roles in the pbk, but for an object you do not assign access rights to these roles, while you do specify that `$REST` should get e.g. read access, then all users will have only read access. However, by default `$REST` has not been set anywhere, and if you do not assign access rights to any roles, then all users will have full access.

A second special role is `&ADMIN` (available from Adlib 6.6.0). This role should be assigned to administrator users, but access rights must never be applied explicitly to this role because users who get the `$ADMIN` role have full access rights by default, plus the right to unlock

manually locked fields and even the right to change the name of the record owner and user access rights per record if record authorisation is active for the database. The `$ADMIN` role and its inherent rights also have priority over access rights assigned to `$REST`. The `$ADMIN` role and its rights even overrule access rights for application roles. The latter means that users with the `$ADMIN` role may get to see an Adlib application in quite a different way, namely with all possible details screens, methods, access points and all possible data sources: this is because in model application 4.2 and higher, application roles are the filter that determines if an object appears in a certain application or not. Since that filter is ignored for an `$ADMIN` user, he or she gets to see all application objects.

In the old application setup tool ADSETUP a roles text file was used for registering roles. However, in Adlib Designer this file is no longer in use. This never causes problems because the combination user-role has always been stored in the `.pbk` file of the application. (The roles file was only used to offer a list of existing roles, when the user created a new user-role combination in ADSETUP. Designer keeps track of existing roles internally.)

Remarks

- User-role combinations may differ per application.
- When a user logs in, he or she is automatically assigned the appropriate role.
- You assign access rights to roles, on the *Access rights* tabs of object properties in the *Application browser*. Do not use the `$ADMIN` role: it has full access rights by default, everywhere in the application.
- If you assign access rights to roles for databases and datasets, these access rights apply to all applications that use these databases.
- Access rights to roles for objects in a database definition (`.inf` file) can never be extended by wider access rights to roles for objects in an application definition (`.pbk` file), only limited.

See also

Accessing the application setup

Managing application objects

Security in Adlib

User properties

3.9 Database aliases (FACS)

FACS stands for File Access Control System and is a subsystem of ADAPL, the Adlib programming language. Using this programming language, databases or datasets are approached using an alias for the actual database *.inf* file name.

This makes it possible, for example, to display a loan status and an order status for a given book in the catalogue at the same time, even though the data is contained in different databases.

So you create database aliases if you want to access databases or datasets from within adapls; these may be stand-alone adapls or adapls that are executed automatically after certain user actions in the running application. Such adapls may be used to only retrieve data or also to write data back.

Database aliases for adapls that are used in an application (not possible for stand-alone adapls) can in principle be specified in the application setup, but this functionality is deprecated: database aliases for both types of adapls should always be specified in the adapls themselves.

See also

Accessing the application setup

Managing application objects

Database alias properties

ADAPL programming: FACS

3.10 Help texts

Adlib users can open context-sensitive help for all screens, list items and all editable fields in the detailed display screens.

Help texts are displayed in a separate window. The standard window can display a large body of text. And if this text is longer than the height of the Help window, the user can scroll through it.

Help texts are created as text files with the extension *.hlp* or *.adh* (the latter from model application version 4.2) in the *\texts* subfolder of your Adlib system. The encoding of a help text file may be MS-DOS, ANSI, or Unicode in UTF-8 representation. All Help files for an application must be saved in the same encoding (you can choose the encoding when you save a text file in a text editor).

Help texts do not contain tooltips, system messages and dialog text. These are contained in system text files with the extension *.txt*. We advise to not change these files, as they will be overwritten with every update of the Adlib software.

[Click here](#) for more information about Adlib file types and their locations.

The help texts can be edited using an appropriate* text editor, such as WordPad (* appropriate for the encoding type of the file). Make sure you only edit the *applic#.hlp/adh* files for the appropriate application. *Adlib.hlp* files should not be altered as they contain non-specific Help for the Adlib software itself, and they are overwritten by every update of the software.

Each help text (i.e. each topic) in a Help text file has to have a unique name, which can be up to 32 characters long, to point to the topic from within the software. This name is known as the help key. In the application setup you define a link between a data source or a method and a Help text, by assigning a help key to the *Help key* property on the concerning properties tab for a selected data source or method; and in the Screen editor you define a link between a screen (and automatically any of the entry objects on it) and a Help text, by assigning the *Help key* property of the screen a help key.

By giving the Help files standardized names, with a number indicating the language it's written in, and putting them in the *\texts* subfolder (or the appropriate application folder for model applications older than 3.3), Adlib knows automatically which files to search for the help keys provided as properties. For instance *applic1.adh* in the *\texts* subfolder of an Adlib system, provides all Dutch Help texts for Adlib Library, Museum and Archive (English = 0, French = 2, German = 3, Arabic = 4, Italian = 5 and Greek = 6). The language of the displayed Help in the Adlib user interface depends on the user interface language set by the user.

If you find that switching between Adlib user interface languages doesn't switch between Help language, you probably have an *applic.hlp/adh* file without language number: this file will always take precedence over *applic#.hlp/adh* files with a number. So make sure your application folders only contain numbered *.hlp* or *.adh* files: before you throw away the *applic.hlp/adh* without language number, check that the translation in this file is present in one of your numbered *applic#.hlp/adh* files as well. For example, if *applic.hlp* contains English text, but you do not have an *applic0.hlp* file yet, then rename *applic.hlp* to *applic0.hlp*. (The same applies to *adlib.hlp* and *adlib#.hlp* files, by the way.)

It is also possible to use one *applic.hlp* or *applic.adh* file (per language) for more than one application simultaneously: see the *Help texts property* in application definitions.

Each Help topic in a Help file begins on a new line and starts with two exclamation marks, immediately followed by the help key. For example:

```
!!DATASET_HELP
```

On the next line is the actual help text for the topic that is displayed to the user. This includes all text (including blank lines) right up to the next occurrence of two exclamation marks. You can enter comments that are not displayed to the user by starting a new line with an asterisk. For example:

```
* The user does not see this line.
```

There are four different types of help keys you can use:

Ed<helpkey><tag>	Field-specific help during edit mode in the detailed display screen in Adlib.
Ed<helpkey>	Screen-specific help in edit mode.
<helpkey><tag>	Field-specific help during display mode in the detailed display screen in Adlib. If the Help window is active, help appears for a specific field when you move the mouse over the field.
<helpkey>	Screen or dataset-specific help in display mode.

When using Adlib, the user can obtain help by pressing F1 (Help). Adlib then selects the text for that screen, list item, or field, etc., as defined in the application setup.

Note that you can't specify a help key in the properties of an entry field on a screen; it is enough to specify a help key in the screen properties, and add tag-specific help keys (as specified in the table above) and topics in the Help file, and Adlib will automatically look for an appropriate field-specific topic when the cursor is in an entry field on a screen or moves over it.

Let's look at an example. If a user has a detailed display screen open and is currently editing a field with the *au* tag and the screen has the help key *BOOKHLP*, then Adlib will look in the *applic.hlp/adh* file for the line: *!!EDBOOKHLPau*. If this is found, Adlib will read the text that follows this line, up until the next line that begins with *!!* and display the text in a Help window. If *!!EDBOOKHLPau* isn't found, then Adlib looks for *!!EDBOOKHLP* (the key for the entire screen in edit mode), and if that isn't present then *!!BOOKHLP* will be searched for (the key for the entire screen in display mode). After that, if still no topic has

been found, the same search sequence will be repeated in the *adlib.hlp/adh* file. And if that still doesn't yield a topic, the default help key for the current Adlib software function will be searched for in the *adlib.hlp/adh* file (if this function has been assigned such a key in the software), so that if, for instance no application-specific Help is specified for a data source in the first step of the *Search wizard*, the default Help for this list in general will be displayed. (You can find out what these default Help topics are, by opening your *adlib.hlp/adh* file in WordPad.)

An example of (part of) an *applic* help text file is the following:

```
!!EDDETAIL_HELPaa
* quantity field
[number]
Enter the number of materials in the set. For a double CD set,
for example, you would enter 2 here.

!!AC_TITLE
* access point title in catalogue datasets.
[title]
Choose this option to search on one or more words in the title
and the subtitle.
```

If you wish, you can use a customized screen with fixed text on it, instead of a help text. When Help is activated by the user (**F1**), Adlib first looks for a screen with the same name as the help key value. If no screen is found with that name, Adlib looks in the help text file for the corresponding help topic and displays that in the standard help window.

If the user presses **F1** or **Enter** and no Help screen or topic is found, a message is displayed, for example: *No help available at this point*. If possible, Adlib will also display the corresponding help key.

3.11 Interface functionality for the application setup

§

3.11.1 Accessing the application setup

To access the application setup for your Adlib application in Adlib Designer, you have to open the *Application browser*. With the *Application browser* you can scroll through your application similar to scrolling through folders and files on your computer with Windows Explorer. But in the application/object browser you'll only see folders

and Adlib objects, like screens and application definitions and databases. From here, you can add new Adlib objects too, to the folders or lists of your choice. And if possible, you'll find the properties of a selected object in the right pane of the *Application browser*, where you may edit them.

Follow these steps:

1. In the main *Adlib Designer* window that opens when you start Designer, start the *Application browser* by choosing *View > Application browser* or clicking the button for this tool:



2. Select your work folder in the *Application browser* window, by clicking the *Open folder* button:



Preferably, choose your main (copy of an) Adlib folder, not one of the subfolders in it. This allows you to browse all your Adlib objects quickly.

3. In the left window pane of the *Application browser*, click the **+** in front of each folder or object to expand the tree structure and display all objects or folders underneath the current item*. Click **-** in front of each folder or object to fold it in. Application management takes mainly place in the folder with the name of the application. (Click here for an overview of Adlib file types and folders.) In this application folder the concerning application definition is present, and maybe also some other files, for instance screen files. Screen files may also appear in other folders. So click the desired application folder node and then select or open the application definition file to access the true application setup part. The properties of a selected Adlib object are displayed in the window pane on the right; some nodes are just list headers and no objects, like *Methods (#)* or *Users (#)*, and therefore have no properties. Underneath an application folder node in the tree view you may see the following Adlib object descriptions:



- an application definition (*.pbk* file) for use with *adlwin.exe*



- an application definition (*.pbk* file) for use with *adloan.exe*



- data sources



- methods



- references to list screens



- incorrect reference (screen file not present)



- references to detail screens



- incorrect reference (screen file not present)



- references to search screens



- incorrect reference (screen file not present)



- output jobs



- export formats



- friendly databases



- users



- library branches (locations)



- Fine tables



- database aliases



- screen files. Of screen files only a few properties are present in the *Application browser*. All properties of a screen can be found while editing the file in the *Screen editor*.

* If your *\data* subfolder is not in the place where Designer expects it (in the same folder that also holds the currently opened Adlib application subfolder), then on opening a data source node in this *.pbk* file, you may encounter an error message notifying you that the database cannot be opened. Then you'll have tell Designer where to look for the *\data* folder. Right-click the application definition node in the tree view, and choose the *Alternative data folder* option at the

bottom of the pop-up menu. In the window that opens, select the `\data` folder for this application. Now you can open the data sources normally. This alternative data folder setting for this application, will be stored in the Windows registry.

Note again that this situation only occurs when your `\data` subfolder is not in the normal place, and in your `adlwin.exe` shortcuts you use the `-datadir` command-line option to point to the other location too.

See also

Managing application objects

Editing properties

Saving modifications

3.11.2 Managing application objects

When you have opened an application folder node of your Adlib application in the *Application browser* (see Accessing the application setup), you can edit the properties of each object, but you can also do some object and file management in the tree view:

Find in application tree

To search for a term in the tree view of the *Application browser*, choose *Edit > Find* (**Ctrl+F**) or click the button for it:



In the *Search for* entry field, type any term or part thereof that you want to search in all of the text displayed in the *Application browser* tree view.

If the term you type is only part of a word or words you look for, then deselect the *Match words* option.

Leave the *Match case* option unmarked if upper and lower case are not important while searching.

Click *Find* to start the search. A found term is highlighted. To search a next appearance of the searched term, each time click *Find next*, press **F3** or click the button for it:



Copying, moving and sorting

Parts of an application in the tree view can be moved by cutting or copying a selection and pasting it elsewhere. Right-click an object and choose *Copy* (**Ctrl+C**) or *Cut* (**Ctrl+X**) in the pop-up menu. Then select another list or folder, right-click it and choose *Paste* (**Ctrl+V**) in the pop-up menu. This way, you can copy an entire application definition or any of the objects in it to a different folder or appropriate object or list. For instance, if you copy a specific method, you can only paste it in another selected data source node, a selected *Methods* list header, or a selected method.

When you paste an object that is part of an application definition, it will never overwrite the other selected object or another object with the same name in that list.

You can also drag objects from one list or folder to another (to where the mouse pointer displays a +), or in the same list. (Dragging means clicking an object, keeping the left mouse button pressed down, and moving the object some place else, and then releasing the mouse button.) Dragging an application object or screen file to another list or folder means copying it. Dragging an application object to another place in the same list, means moving it; this is relevant for the data sources list, the *Methods* list, and the *Output jobs* list, because the order of these lists is the order in which they are presented to the user in the running application.

The files in a folder node (not the objects in an application definition) can be sorted alphabetically: in a typical application folder, this means sorting the *.pbk* file and the screen files. Just right-click the folder node and choose *Sort* in the pop-up menu.

Creating new objects

New items can be created. In the tree structure, select the folder or the object in which you want to create a new folder or new Adlib object, and create the object through *File > New*, or right-click the folder or object and choose the new object through the *New* option in the pop-up menu. What objects are available in the *New* menu depends on the node that you selected. For instance, to create a new data source, user, or database alias in an application definition, you must select or right-click or select the application definition name, not a specific data source or folder. But from a specific data source node, you do create new methods, references to screens, output jobs, export formats, or references to friendly databases. In the *Application browser*, new screen files can only be created from a folder node.

Creating documentation

In some Designer tools, like in the *Application browser* for a selected database or application, or in the *Import* or *Export job editor*, or the

Record lock manager, it's possible to generate documentation about the currently selected object or the list of objects, by choosing *File > Create documentation* or clicking the button for it:



A *Documentation* window will be opened with a detailed description of the structure of the selected object. For a database for instance, this information comprises data on the database, its datasets, fields, indexes and links. And the documentation for an application will contain a detailed overview of its properties, menu texts and data sources.

The overview is nicely laid out, but if you rather have the bare XML view, you can switch to it through the *View* menu.

In either view you can print the file via the menu or the *Print documentation* button, or save it as an XML file.

You may also select the text, or part of it, with the mouse cursor, and copy (**Ctrl+C**) and paste it (**Ctrl+V**) in any text editor document.

Deleting objects

Select any node in the tree view (a folder, an application definition, a screen file, a method, an output job, etc.) that you want to delete and either choose *Edit > Delete*, or right-click the object and choose *Delete* in the pop-up menu, press the **Delete** button on your keyboard, or click the *Delete* button in the toolbar:



Note that deleting any object that has sub-objects, also removes those sub-objects. So, removing for instance a data source, also removes all methods, screen references, output jobs, etc. in that data source! Folders are deleted along with their contents too. Deletion of files is currently permanent: you cannot restore deleted files from Windows' recycle bin. Deleted sub-objects within the application definition, can be restored by not saving the changes in the concerning *.pbk* file when you close Designer.

See also

Editing properties

Saving modifications

3.11.3 Editing application object properties

If properties of an object are displayed in white or yellow entry fields, you can edit them. (The yellow colour is just for visual presentation, it has no special meaning.)

Titles	
Language	Text
English	ADLIB Library 2.1.1
Dutch	ADLIB Bibliotheek 2.1.1
French	
German	ADLIB Bibliotheek 2.1.1
Custom1	
Custom2	

Just click in the entry field, and delete or type characters.

Database	<input type="text" value="document"/>	<input type="button" value="..."/>
----------	---------------------------------------	------------------------------------

If properties of an object are displayed in greyed out entry fields, you cannot edit them: they are read-only.

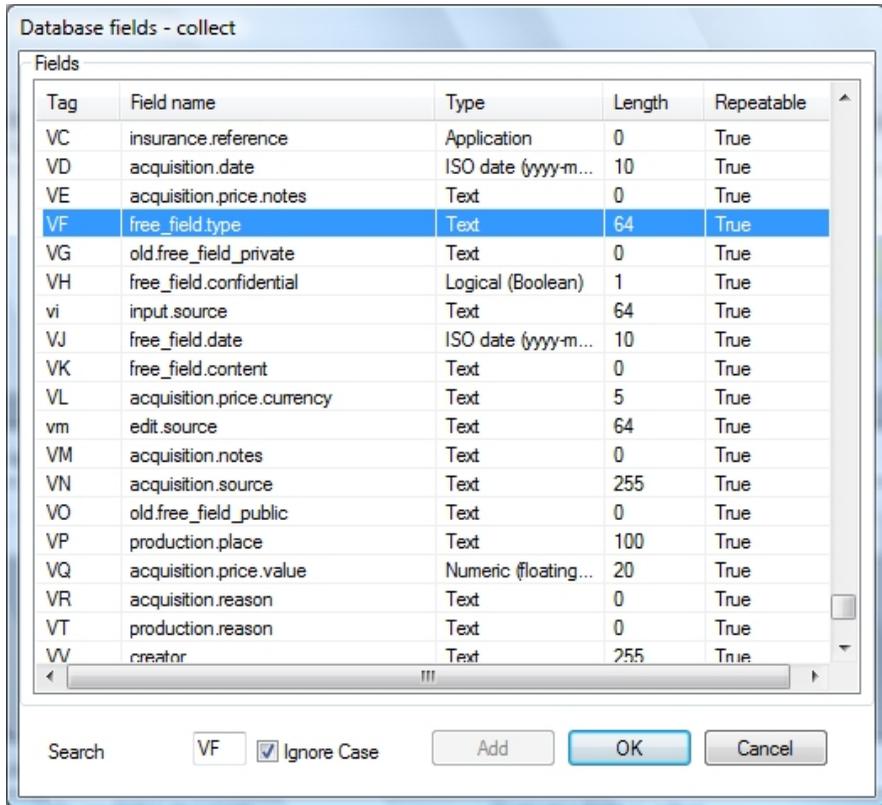
Sometimes you can only choose values from a drop-down list. If such a property reads "*Undefined*", it means that for that property the default value will be used: usually the default value is the first value in the drop-down list.

Choosing a value from a list

Behind a number of entry fields, you'll see a grey button with three dots on it (see the figure above). Click this button to either open a list of files/objects that are available for the current property, to search your system for a specific file/object, or to create a new file/object at a specific location on your system. Whenever the button is available, you are advised to use it. For properties that need a tag or field or other object name (like a dataset name), the list offers all the objects that you can enter for this property, and choosing one from the list makes sure the name is spelled just the right way and that any path to the object is entered correctly.

Even though when you type values yourself, sometimes they are validated against the list that you open with the ... button, there is still a chance that by typing you enter incorrect values, either because you forget that certain values are case-sensitive or because you remembered the name of a tag incorrectly. So use the ... button whenever possible. It helps enormously to be able to just choose from the values that apply for the current property. For example, for

the properties of a data source you can choose the associated *Database* name from a list of database files, and then for the properties of a method in that data source you can choose the *Search field* from a list of fields available in that associated database! In the list window that opens when you click the ... button to search for a database field, you choose a field either by double-clicking it, or by selecting it and clicking the *OK* button (not visible in the reduced image of that window, below).



You can sort the list on any column ascending or descending by clicking the column header once or twice.

You can limit the list to all tags that begin with a certain character by typing that character in the *Search* entry field. Remove it to display the entire list again. If you mark *Ignore case*, the tags searched with the character(s) that you provided can be upper case as well as lower case.

If in the *Search* field you type a tag that does not exist yet, and you click *Add*, the new tag is forced in the field property, but no field

definition is created in the concerning database, so use this option sparsely, if at all.

Change the name of an object

If you change the name of an object through its properties, the new name will also appear in the current node in the tree view to the left of the properties.

The other way around also works: if you select the name of an object in the tree view, and then click it again or press **F2**, you can edit the name in place. This is reflected immediately in the properties of the object. Sometimes you can change an object name this way, when the object properties show the name as read-only.

But be careful with changing names of properties. Currently, Adlib Designer does not do anything with the new name, even when there are many objects that refer to this object. So you'll have to find all references to the old name of the object in other objects and change them manually. In the future, Adlib Designer may do all this renaming of references automatically.

Moving through properties

You can easily move downwards through properties by pressing **Tab** on your keyboard, or **Shift-Tab** to move upwards.

Editing mappings

On some properties tabs, like *Method properties* or *Access rights*, you'll find mappings of properties or objects, like assignments of access rights to roles, or of sort criteria to fields.

The appearance of mappings varies with the version of Designer and with the requirements of the mapping:

- In older versions of Designer (6.3.0 and older) you may see mappings accompanied by a button column to their right side: click the *Add* button to add a new entry in the mapping list; click *Delete* or *Remove* to delete the currently selected mapping entry; if *Up* and *Down* buttons are presents (for e.g. an enumeration mapping) you sort the mapping with these buttons.

Access

	Role	Access rights
	system administrator	Full
	co-worker	Write
...	student	None
*		Undefined
		None
		Read
		Write
		Full

Buttons: Add, Delete, Delete all, Copy, Paste, Roles

- From 6.5.0, you may encounter mappings or lists without *Add* or *Remove* buttons next to it, like in the figure below. These mappings always have a new empty line present, indicated by an asterisk in front of it: as soon as you enter a value, a second new line will be created automatically. To enter a value, just click the relevant entry field and start typing. If you click the asterisk in front of the line, some default values (if applicable) will be entered for you (which of course may still be changed).

	Date	Opening time	Closing time	Return time
*				

To delete a line (other than a new line) if no *Remove* button is present, just click the grey box at the beginning of that line: this selects the entire line. Now press the **Delete** button on your keyboard to remove the line.

	Day	Amount
	1	50
	10	100
	20	150
▶	30	200
*		

If no buttons are present for a mapping or list, all functionality for the relevant mapping is available in a pop-up menu which opens as soon as you right-click a line in the mapping.

When adding a new item or editing an existing entry, you sometimes

have to choose options from drop-down lists that appear as soon as you select the property or object in the mapping, or you must choose an object from an objects list in a new window. In general, click the property or object in the mapping to edit it, and you'll be presented with the available choices. Sometimes you may also type a new value. See the Help topic for the concerning property for more information.

Managing roles

On the different *Access rights* tabs in Adlib Designer 6.5.0 or higher, you'll find the *Roles* option in the pop-up menu: right-click a line in the mapping to open the pop-up menu. Choose that option to open the *Current list of roles* window. You are presented with an overview of all roles which are currently in use in your Adlib system. Adding a new role from the *Access rights* tab can only be done in this window: just click the empty entry field and type the name of the new role. Now it is important that you move the cursor to the next empty line after you've entered the new role, otherwise the new role won't be stored in memory. Close the window via the x button in the title bar. The new role can be found in the drop-down list which opens when you want to select a role to assign access rights to.

Note that the new role will only be saved if you actually apply it somewhere in Adlib, to assign access rights to an object. Moreover, the overview in *Current list of roles* will only be refreshed (put together again) when you restart Adlib Designer: redundant empty lines or roles which are no longer in use, will automatically disappear from the list. So you don't need, and can't delete roles manually: a role will be removed from the lists when it is no longer used.

Copying access rights

From Designer 6.5.0, roles and assigned rights set up for one Adlib object, can be copied and pasted to other Adlib objects, even to multiple objects at once. Moreover, copied roles and rights can be edited on a special clipboard, before you paste the access rights to other objects.

Proceed as follows to copy access rights from one Adlib object to others. First, right-click any of the lines with assigned access rights on the *Access rights* properties tab for the relevant Adlib object, and choose *Copy* in the pop-up menu which opens. This copies all access rights to the *Access rights clipboard*, which is really a skin around the Windows clipboard. You can view its content and edit it if you wish. Choose *View > View rights clipboard* in the menu to open the special clipboard. If you are satisfied with these access rights, then close the clipboard via the red-white cross in its title bar. However, you can still make changes first. Just like on the *Access rights* tab, you may edit

roles and rights, add new rights through the *Add* in the pop-up menu, delete a selected line with *Delete* or delete all copied rights via *Delete all*, also from within the pop-up menu. After closing the *Access rights clipboard viewer* (if you had opened it), you may paste the copied access rights to other objects in several different ways:

- paste to a single object: open the *Access rights* tab of the other object, right-click the list and choose *Paste* in the pop-up menu. Or right-click the object in the tree structure of your Adlib system in the left pane of the *Application browser*, and choose *Paste access rights* in the pop-up menu. Or right-click the object in a list in the right window pane of the *Application browser*, for instance a field list, and choose *Paste access rights* in the pop-up menu. Observe that access rights may also be copied or deleted from within such a list, via the pop-up menu.

- paste to multiple objects at once: in an objects list in the right window pane of the *Application browser*, you can select multiple objects. Keep **ctrl** pressed down while clicking all objects you want to select, or click the first object and keep **shift** pressed down while you click another object to select all in between objects at once. Then right-click one of the selected objects and choose *Paste access rights* in the pop-up menu to apply the copied access rights to all selected objects.

Note that all existing access rights of the target object will be overwritten when pasting!

By the way, it is not necessary per se to copy an object's access rights first. You may also begin by opening an empty *Access rights clipboard viewer*, and then add all desired access rights via *Add* in the pop-up menu.

Editing screens

In Adlib Designer the former ADSETUP functionality has been split up in two: setting up an application, and designing screens. Designing and editing the layout and properties of screen files and the objects on them, is done in the *Screen editor*. (A small selection of the properties of a screen file can also be edited in the *Application browser*.) Double-click a screen file or screen reference in the tree view of the *Application browser* to edit the screen file in the *Screen editor*.

See also

Accessing the application setup

Managing application objects

Saving modifications

3.11.4 Saving modifications

Changes* in the properties of an object in the application setup can be saved directly manually in the current application definition (*.pbk* file) by right-clicking the application node which the object is part of, and choosing *Save* in the pop-up menu, by choosing *Save (Ctrl+S)* in the *File* menu or by clicking the *Save* button:



Since all sub-objects in an application are stored in one and the same *.pbk* file, you cannot store changes in the sub-object separately; you have to save the application structure as a whole to save the changes to an individual method, for instance.

You can also only save changes in selected files, by choosing *File > Save all...* (*Ctrl+Shift+S*) or by clicking the *Save all* button:



In the *Save objects* window that opens when you click this button, you will be presented with an overview of all the unsaved files in which you made changes, and you can determine of each of these separately whether you want to save them or not, by selecting them or deselecting them. Click *Yes* to save the selected files. *Cancel* returns you to Designer.

When you close Adlib Designer while you haven't saved all changes yet, the *Save objects* window will automatically appear, allowing you to save your work before the application is closed.

Note that in the *Application browser* only physical Adlib objects (files) can be saved separately (currently these are just the *.pbk*, *.inf*, and *.fmt* files). This is because sub-objects like e.g. data sources or methods, are not single files. To save changes in sub-objects immediately, save the larger file that incorporates them, like the application definition.

While managing objects in the tree view of the *Application browser*, there are a few actions that initiate an automatic save of the concerning object, namely the creation of an application definition, a screen file, or folder.

* The nature of properties forms and lists sometimes requires you to leave a property that you just edited, before you can save it. So when you have edited your last property for today, make sure you first click some other property or tab, before you save all your work.

Your .pbk files are safe

To prevent possible loss of *.pbk* files because of any errors occurring while you save an *.pbk* file, there is an automatic backup procedure in place. When you instruct Designer to save your work, it first makes a backup of the files to be saved by renaming the original files (meaning: the files in which you have not yet saved your current changes). Those backup files have the same name but the extension *.bak*. In the unlikely event that the subsequent saving of the changed *.pbk* file corrupts the file, this file will automatically be closed and deleted, and the *.bak* file is given back its original extension *.pbk*. Of course you must then enter your changes to the file again, and try to save it more successfully; but at least you will have a proper *.pbk* file. But usually saving your files will happen without problems, and when a save has indeed been successful, the automatically created *.bak* files will be deleted automatically again too, as if they were never there.

3.12 Properties of Adlwin application objects

§

3.12.1 Application properties

On the *Application properties* tab, which is present when you have selected a new or existing application in the *Application browser*, you determine the general properties of this application.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Path

An application structure is stored in a *.pbk* file. Each application or module is usually saved in a folder of its own. The full path to the currently selected application structure is displayed here.

For safety, only edit a copy of your live application, and test it well before using it.

If you want to change the location of this application, use the functionality in the tree view or in Windows Explorer to move files from one folder to another, or to rename the folder.

Encoding

This read-only property displays the type of character set used to encode texts you provide for properties of this object. It does not say anything about the character set used to encode data in. You can change the encoding of database structures (*.inf* files), application structures (*.pbk* files) and screens (*.fmt* files) simultaneously, with the *Application character set conversion tool* in Adlib Designer.

Identification

Here you may provide an identifying name for the current application, with the purpose of implicitly creating an "application" role with this name. But you must not assign users to this role, because this role automatically applies to everyone using this application! This application role can be assigned access rights for specific objects of the application or for database objects. So, if you were to define an *Identification* for this application, and then map this role to specific access rights for, let's say, a screen file, then you're saying: if this screen file is accessed in this application then these access rights should apply; for use in other applications these access rights do not apply. This way, for instance a standard screen may be read-only in one application, whilst it may be editable in another application.

In the old DBSETUP program all roles, including application roles had to be included in the roles file before you could use them to apply access rights to application or database objects. This was just a text file without extension, named *roles*, that had to be located in your *\tools* folder (or the *\bin* folder in absence of a *\tools* folder), and in which you summed up the names of different roles, for example:

```
students
teachers
public
public access application
management application
```

In Designer, the roles file is no longer used: it is enough to specify roles for *users* of the application.

So, whenever you wish to apply access rights to an application object or database object, you can choose between all user and application roles specified in the application setup.

Since access rights assigned to user roles and application roles can be contradictory, the most restrictive rights of the two, in any situation, apply.

Use large buttons on the toolbar

(Not applicable to the ribbon in Adlib 7 or higher.)

- *True* - Buttons appear extra large on the toolbar of a running application.
- *False* - Buttons appear on the toolbar in their normal size.

Use screen color on tabs

Normally in a running application running on Adlib 7 or higher, the outer border and the label of a tab are light blue, and labels turn dark blue when their screens aren't active. The main body of the screen may have another colour: in old model applications this used to be grey while in model applications 4.2 this is light yellow by default or any other colour chosen by the application manager.

Mark the current option if you would like the outer border and label of all tabs (active and non-active) to be presented in the same colour as the main body of the screens.

Application background colour

You can set the background colour of a whole application at once. This concerns the colour of the window pane **in** which the *Search wizard* appears, for instance, and the tab sheets of a detailed display. You will only see that colour when a small window is displayed in that larger window pane (*Search wizard*, *Pointer files* and *Expert search system*); when tab sheets are being displayed, the background of said window pane is not visible.

Click the coloured box to open a standard Windows colour picker. You can select a basic colour, or click *Define Custom Colours* to choose another colour.

Click here for more information about changing the colours of your application, screens and screen elements.

Allow searches on empty keys

- *True* - It is allowed to search on empty keys in text (term) indexes.
- *False* - The user must always enter a key.

Truncate free text searches by default

- *True* - Keys used for searching in free text indexes will be regarded as truncated keys. To search without truncation, the user must embed the key in double quotes.

- *False* - Keys used for searching in free text indexes will be regarded as the key (i.e. the full text). The user must type an asterisk behind an entered key to search truncated.

Enable the ALL KEYS button

- *True* - In the *Search wizard* in a running application the *All keys* button will be active when it displays all found keys after a search. If the user clicks this button, the records for all these keys will be retrieved.
- *False* - In the *Search wizard* the *All keys* button will not be active. The user can therefore not search for the records of all keys at once, but must click a key, after which only records for that key will be retrieved.

Allow Boolean searches in the Search wizard

- *True* - The user may combine searches using the Boolean operators (AND, OR and NOT).
- *False* - The user cannot combine searches using the Boolean operators.

Maximum number of keys displayed

This value (by default 5000) indicates the maximum number of keys initially displayed after a search.

Maximum number of retrieved records

This value (by default 5000) indicates the maximum number of records initially displayed after a search.

Milestone value

This value (by default 100) indicates the interval at which various (search) counters on the screen are incremented. (Counting every searched record may slow down the search.)

Allow printing to file

- *True* - The user can print to file.
- *False* - The user cannot print to file.

Allow on-screen print previews

- *True* - The user can display a print preview on screen.

- *False* - The user cannot display print previews.

Allow printing to printer

- *True* - The user can print to a printer.
- *False* - The user cannot print to a printer.

Allow printer selection by the user

- *True* - The user is allowed to choose a printer prior to printing.
- *False* - The user cannot choose a printer.

Limit the number of printed records at

Specify the maximum number of records that is allowed to be send to the printer at once. To permit an unlimited amount of records to be printed, set this property to 0.

Default access rights

Set here the standard access rights for this application, that must apply if no access rights have been set for an object through the roles functionality. If a role and access rights have been mapped for any application object or database object, then those access rights always have priority over the *Default access rights* for the application (whether those role-specific access rights allow more or fewer user actions than this default). By default, the *Default access rights* for an application are set to *Full*.

3.12.2 Application titles

On the *Application titles* tab, which is present when you have selected a new or existing application in the *Application browser*, you determine the title of this application as it should appear in the title bar of a running application, in all the languages you want your application to be available in.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Titles

You should provide the title of this application in all the languages in which the user must be able to work in the running application. A title will appear in the title bar of an Adlib application window.

3.12.3 Authentication

On the *Authentication* tab, which is present when you have selected a new or existing application in the *Application browser*, you determine whether you want to set a login procedure for opening your Adlib application. This means that every user that opens the Adlib application will first have to log in with his or her user name and password. This extends the Windows user authentication. See the Help topic: *User authentication and access rights*, for more information.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Authentication method

- **None** - Adlib will (implicitly) use the user name from the general Windows login for applying any security policy through access rights and roles. In Adlib, users must still be linked to roles, which can be done in the application structure (*.pbk*). No login window will appear when starting Adlib.

If you want to set an explicit application login procedure - an Adlib login window will appear when starting Adlib - then choose the type of authentication you want to use:

- **Adlib.pbk** - store user names, passwords and roles in a *.pbk* file through user properties editable in Designer. The user names may be different from user names for the general Windows login.
- **Adlib database** - store user names, passwords, roles (optionally) and application ids (optionally) in one of your Adlib databases. You'll probably have to adjust your application to make it possible to enter this information in the database. You'll then have to fill in the other options on the current properties tab. The user names may be different from user names for the general Windows login.
- **Active directory** - store user names and passwords in Active directory user accounts.
- **HTTP** - register user names and passwords through some remote service (typically no Adlib service).

See: *User authentication and access rights*, for more information about all authentication methods.

Folder

Only if you've selected the *Authentication method: Adlib database*, this property must contain the full path to the database you want to link

to, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the path to that folder is automatically entered here.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that holds all the user names and passwords (only for the *Authentication method: Adlib database*). Do not enter the extension of the file. An example of such a database name is PEOPLE.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like *Persons and institutions* (PEOPLE), have no datasets.) Selecting a dataset is optional, you can also just link to an entire database. Typically, you select a dataset if for this link you only want to retrieve data from that specific dataset.

User Id field

In the above chosen database, select the (existing) data dictionary field that holds all user names.

See: User authentication and access rights, for more information.

Password field

In the above chosen database, select the (existing) data dictionary field that holds all passwords.

See: User authentication and access rights, for more information.

User role field

In the above chosen database, select the (existing) data dictionary field that holds all user roles.

See: User authentication and access rights, for more information.

Application Id field

In the above chosen database, select the (existing) data dictionary field that holds all application ids.

An application id field is only necessary if users must have different roles in different applications. The user role and application id fields must then be grouped and repeatable.

See: User authentication and access rights, for more information.

Format string

If you've selected the HTTP authentication method, you need to

provide a format string containing a fixed URL to the remote verification service, and two placeholders for the current user name and password to submit to the remote service.

On starting this Adlib application, an Adlib login window will appear; the user name and password entered in there, will be copied to the placeholders by the Adlib software, and submitted to the remote service automatically.

The placeholders in the *Authentication format string* are literally indicated as follows:

`%username%`

`%password%`

For the complete URL, you enter the actual URL (CGI string) to the remote service that must handle verification of the user name and password contained in this string.

Fictive examples of such URLs are the following:

```
http://ourremoteserver/subdirx/webservice.asmx/Authenticate?
userName=%username%&userPassword=%password%
```

```
http://demo.adlibsoft.com/samples/validateUser.aspx?UserName=%
username%&Password=%password%
```

```
http://users.adlibsoft.com/checkUser.aspx?p=%password%&u=%
username%
```

Note that the exact syntax of the URL is determined by the implementation of the authentication script on the server.

3.12.4 Advanced

On the *Advanced* tab, which is present when you have selected a new or existing application in the *Application browser*, you determine the more rarely used properties of this application.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Skip list screen if result contains 1 record

- *True* (select this checkbox) - The brief display will only be shown if more than 1 record is found after a search.
- *False* (leave this checkbox unmarked) - The brief display is always shown, even if only 1 record is found.

Combine search display

- *True* (mark this checkbox) - During combined searches (with Boolean operators) in a running application, a list of matching keys is displayed after the second and subsequent searches, like for any search. In this list, you can select a key or keys to combine with the previous search.
So for instance, after searching on a term with the *Search wizard* the user ends up with a brief display. Then the user can combine this result with a new search by clicking the *AND*, *OR*, or *NOT* button. Now the new search will be combined with the result you already have. And with the current application property, you determine whether when the user specifies a new term to search for, the user will be able to choose a key from a list first (before retrieving records for it), or not. Select his property to allow the user to make this choice.
- *False* - During combined searches, no list of found keys will be displayed after the second and subsequent searches. All the keys found are combined with the first search, to retrieve a list of records.

Sort the data source list in the Search wizard

- *True* - The data sources listed in the first step of the *Search wizard* in a running application) will be displayed in alphabetical order.
- *False* - The data sources listed in the *Search wizard* will be displayed in the order in which they are listed in the tree view in Designer under the application node. (You can change that order by dragging a data source to another place in the data sources list in the tree view.)

Return to first tab when moving between records

This option is used when browsing the results of a search, that consists of several records, and when there are two or more detailed presentation screens (tab sheets in the running application) available for each record. It determines whether from every subsequently opened record, the same tab as the current should be displayed, or the first tab.

- *True* - When the user moves to the previous or next record in the search result, the first tab of that record presentation will be shown.
- *False* - When moving to the previous or next record in the selection, the currently opened tab, but from the other record, will be displayed first.

Help window visible by default

- *True* - As soon as Adlib has started, the Help window will be displayed.
- *False* - Help will not be displayed by default; the user has to open it manually in the application (F1) when it is needed.

Adlib menu style (obsolete functionality)



From Designer 7.1.0, this option is not present anymore and the dashboard functionality is no longer supported by Adlib, meaning that earlier set up dashboards will be ignored. So the following text does not apply to Adlib 7.1.0 and higher.

Enter the (preferably relative) path to a stylesheet, if you want this application to use that stylesheet to lay out the data source list in non-classic display mode. An example of a relative path would be: . .

`\dashboard\DashboardXplus.xsl`

(Click here for the general topic about this functionality.)

Show "help" and "return" methods

This option only applies to the DOS version of Adlib and need not be set for Adlib Windows applications.

- *True* - The *Return* and *Help* options are available for the database and access points lists.
- *False* - The *Return* and *Help* options are not available for the database and access points lists.

Show the main menu

(Not applicable to the ribbon in Adlib 7 or higher.)

- *True* - The menu bar will be displayed in the running application.
- *False* - No menu bar will be displayed. (This option is used in the Library OPAC, for example.)

Allow the user to shut Adlib

- *True* - The user may close the application.
- *False* - The user is not allowed to close the application. This setting is recommended for terminals for public use, like the Adlib Library OPAC. The only way to still close the application would be via the Windows Task Manager or by shutting down the computer.

Leave the welcome screen by pressing any key

This option only applies to the DOS version of Adlib and need not be set for Adlib Windows applications.

- *True* - After the welcome screen, you can start the application using any key, except for **Enter**.
- *False* - After the welcome screen, you can only start the application by pressing **Enter**.

Time-out value (in sec.)

This value determines the number of seconds of inactivity after which Adlib must automatically return to the first step in the *Search wizard*. If this value is set to 0, Adlib will never automatically return to the first step in the *Search wizard*.

Use this option e.g. for terminals for public use, like the Adlib Library OPAC.

Startup language

Here, choose the language you want the currently edited Adlib application to start in.

For other purposes it is convenient to know that certain fixed numbers are associated to the first four languages, namely:

- 0 - *English*
- 1 - *Dutch*
- 2 - *French*
- 3 - *German*

Encoding of application texts & Encoding of help texts

The encoding of application texts (system texts) and help texts should be seen separately from the encoding properties of application definitions (*.pbk*), database definitions (*.inf*) and screens (*.fmt*), because system and help texts are located in their own files (*.txt* and *.hlp/adh*), and their encoding can be different from the encoding of the application definition. Their encoding must be indicated here, because Adlib needs to know how to represent the characters in these texts on the screen (this is not self-evident to the software).

For existing Adlib applications you only need to change the selected character sets for these options, if you change the character set in

which the text or help files are saved. When you edit these files in e.g. Notepad, you can choose *Save as* to change the name of the file and/or the character set. Typically you will only change this, if in the text you want to add to these files, characters appear that are not supported by the current character set. Do make sure all text files for this application are saved in the same character set, and that all help files are saved in the same character set too.

These options replace the *Convert application/help texts to ISO* options, and offer the following character sets to choose from: *DOS (Western Europe)*, *ANSI (ISO-Latin)*, and *Unicode (UTF-8)*.

DOS is the elementary character set, that cannot represent the € character for instance, ISO-Latin does contain most characters used in western European countries, like the euro-sign; if you need to use more exotic characters, like from Chinese or Hebrew for example, you need to use Unicode in UTF-8 representation.

Data languages

This option only applies to SQL and Oracle databases. Fields which you make multilingual in such databases, will be (optionally) multilingual in all the languages which you mark in the current list. (You can always change the selection of offered languages here, without serious consequences.) The user can choose from these languages via the *Data language* button in the running application, when the cursor is in a multilingual field.

(Note that there are two ways of implementing multilingual fields; [click here](#) for more information.)

Adapl

It is possible to put all texts used by adapls in one text file (per language), and open the right language variant in the running application automatically, whenever an adapl needs to read from a text file, by just providing the name of this *.txt* file once, here in the application setup. You provide it without indicating the language number in the file.

So, if you have two language text files named *adapltxts1.txt* and *adapltxts3.txt*, the name to enter for this option is: `adapltxts.txt`.

Adlwin will automatically pick the right language file.

The advantage of this option is that you no longer have to open specific text files in adapls before you can read from them. You can just always read from the text file specified here in the application setup.

Help texts

Here you can provide a (relative) path to an application-specific Help file. By default Help files are sought in the current application directory, but if you don't want that, you can use this option. Specify the Help file without a language number. Adlwin will automatically select the right language file when the user opens the Help.

3.12.5 Data sources

3.12.5.1 Data source properties

On the *Data source properties* tab, which is present when you have selected a new or existing data source in an application in the *Application browser*, you determine the general properties of this data source.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Data source type

The current data source will appear in step 1 of the *Search wizard* in a running application. Typically, a data source will be a database or a dataset, but you may also make this "data source" a link to start up other software from this step 1 (for instance a spreadsheet or a document in a text editor), or to start a stand-alone ADAPL program. For this purpose, the data source type can be *Normal database*, *Shell command* or *ADAPL command*. (Depending on the value you choose, the following options will be different.)

Folder

If you selected *Normal database* as the *Data source type*, the *Folder* option is present.

This property contains the full or relative path to the database you want to link to, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the relative path to that folder is automatically entered here. Usually, you keep your databases in one folder, and the relative path you'll often encounter here is: `../data`.

Database & Dataset

If you selected *Normal database* as the *Data source type*, the *Database* and *Dataset* options are present.

First, enter or search for the name of the existing database (an *.inf* file) that you want to link to. Do not enter the extension of the file. Examples of database names are `DOCUMENT`, `COPIES`, and `Collect`.

Important: when you work in a copy of your live application, then make sure you search the right folder (in the copy) for the proper file: otherwise the relative path will be incorrect when you place back this copy as your live application later on.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) Typically, you select a dataset if for this link you only want to retrieve data from that specific dataset, and if you want new linked records to be created in that dataset automatically.

Selecting a dataset (if available) is mandatory if you want the user to be able to write records to it. If all you need is read-access, you can also link to an entire database by not selecting a dataset; you need not set the *Access rights* for this data source to *Read*, because you should just never define a *Create new record* method for such a data source. (In some existing applications, in the database setup for such a database a "dataset" is defined that covers the entire database or a range of smaller datasets, the purpose being to exclude one or more datasets at the end of the database from being searched by means of access points, and these excluded datasets then also won't appear in the database list in step 1 of the *Search wizard*. This construction is rarely put to use though.)

The hierarchy you see in the database list in a running application, is created automatically: data sources that fall within a larger data source (in range of record numbers) are indented with respect to the larger data source. (You do have to place the data sources in the desired order in the tree view in the *Application browser*.)

Command

If you selected *Shell command* as the *Data source type*, the *Command* option is present.

Type in the path to the desired executable program (or any DOS command), as you would at the system prompt (in a command line window), except any arguments; the user can start programs or batch files from this link in the first step of the *Search wizard*, or delete files, etc. Example: `C:\Program Files\Microsoft Office\Office\excel.exe`

Command arguments

If you selected *Shell command* as the *Data source type*, the *Command arguments* option is present.

If the program or command that you have entered in the *Command* option, needs arguments, then you enter those here, separated by spaces.

Adapl procedure

If you selected *Adapl command* as the *Data source type*, the *Adapl procedure* option is present.

Click the ... button to select an *adapl* on your system that can be executed as stand-alone program. If you type a path and name, enter the name without extension.

Menu texts

Here, you must provide the name of the data source or the program to be started in one or as many languages as you wish to make available to users. This name will be presented to Adlib users in the first step of the *Search wizard*.

Help Key

Enter the key under which the Help text for this data source can be found in the *.hlp/.adh* files for this application. Adlib will use this key (in the Help file preceded by two exclamation marks: **!!**) to locate the Help text. If the user presses **F1** (Help), the Help text for this data source in the database list in step 1 of the *Search wizard*, will be displayed. Example: `DS_FULLLCATALOGUE.`

3.12.5.2 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing data source in an application in the *Application browser*, you can restrict access to this data source, dependent on the user (login name in Windows) and its assigned role. Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this data source. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply.

The access rights that you assign to a role for a data source must be at least as extensive as the most extensive rights to the same role for a method.

Any restrictions imposed here are additional/complementary to the restrictions defined in the database setup. Users who have had their

access restricted in the database setup will therefore not be afforded greater levels of access through the application setup.

If a role is not linked to this data source, then each user linked to that role has full access by default. If no role is linked to the data source, every user linked to a role has full access. A user without a role always has full access. Users are assigned to roles in the application setup.

A role that has *None* access rights applied to a data source, means that users with this role don't get to see this data source in step 1 of the *Search wizard* in the application; they can work with the rest of the application though.

A user that has read-access to a data source can search and display records, but cannot edit them or create new records in this data source.

Do note that applying limiting access rights on data source level may not be as save as applying them on database level. For instance, excluding users from writing in a data source, still allows them to create or edit linked records in it from another data source to which they do have write-access! If you want to make it impossible for certain users to create or edit records, you are better off setting these access rights in the database setup.

See also

Security in Adlib

3.12.5.3 Methods

3.12.5.3.1 Method properties

On the *Method properties* tab, which is present when you have selected a new or existing method in a data source in an application in the *Application browser*, you set the general properties of this method.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Method type

Open the drop-down list to choose a type for this method. For all types applies that you can exclude certain users from the functionality of that type, by setting limiting access rights for this method or for the database or dataset. The following types are supported:

- *Term search* - Create an access point to search a text (term)

index, or an integer, date or logical index.

- *Free text search* - Create an access point to search a Free text index (also called a word index).
- *Create new records* - Enable the possibility of creating new records in this data source: a *New (record)* button will be present.
- *Expert search* - Enable the possibility to search this data source using the *Expert search language*: an *Expert search* button will be present after the user has selected a data source to work with.
- *Query by Form* - Enable the possibility to search this data source using a QBF search form: a *Query by form* button will be present after the user has selected a data source to work with.
- *Delete records* - Enable the possibility of deleting records from this data source.
- *Pointer files* - Enable working with pointer files in this data source. In contrast to the other method types however, you don't need to create a method of this type if all users are allowed full access to pointer files in the current data source: this functionality is active by default, even without the method being present. That is why you won't find a method of this type in most standard Adlib applications (before application version 3.6 anyway). So typically you only specify a method of this type if you want to restrict access to this functionality for some users. In this case, you can apply access rights specific to this method and/or apply default pointer file access rights per database: the access rights per method can be seen as an (overriding) refinement to any general database wide pointer file access rights.
- *Global Update* - Enable the search-and-replace function in a data source, which activates the *Replace in record* button. Search-and-replace allows the user to semi-automatically replace one value by another, in one and the same field in all marked records.
- *Print records* - Enable the possibility to print selected records from this data source.
- *Export records* - Enable the possibility to export records from this data source, using the *Export* option in the running Adlib application.
- *Import records* - Enable the possibility to import records into this data source, using the *Import* option in the running Adlib application.
- *Derive records* - Enable the possibility to derive records from "friendly" databases.

- Date range search* - This method type only applies to applications running on SQL or Oracle databases, and allows for an access point to search two indexes for different date fields simultaneously on a date range. For instance, when you have two date fields, *Date (early)* and *Date (late)*, e.g. for the dating of a museum object, and you have an index for each of these fields, you can introduce this method in your application. A method of the *Date range search* type requires you to provide the first field in the *From search field* method property (see below) and the second field for the *To search field* option. In the running application you now get an extra access point in the current data source, with which the user can search on a date range. (Also see the Date completion option for date indexes.) Besides the *Date range search* method, there's the Named range search method, which takes date range searching a step further by allowing users to use period names to perform date range searches.

Note that "normal" date range access points search only one index on a single date field, and have *Term search* as the *Method type*.
- Link update* - The *Link update* method enables the *Thesaurus update* button/function in a data source of the running application. A thesaurus update lets you manually update specified fields in all marked records in the current data source with the aid of a different thesaurus. This means that any non-preferred terms in those fields will be replaced by preferred terms (according to their specification in said thesaurus).
- Fixed query* - This method type enables users to execute a fixed or partially variable, complex search statement by simply clicking an access point and maybe entering a search key. This is a handy feature to offer users more convenient ways of searching, for example to be able to request a list of recent accessions, or to search for all object records pertaining that artist with the difficult name who is the focus of a lot of attention at that moment. For a *Fixed query* method, you have to fill in a search statement in the *Query* entry field. The syntax of these search statements is largely the same as for those in the *Expert search system*. The name you give to this method will appear in the access points list (*Step 2* in the *Search wizard*) after you have restarted the relevant Adlib application. If the user double-clicks it, the search statement which you set for it will be executed immediately; if the search statement contains a variable, the user is asked to enter a search key before the actual search begins. If you want to add one or more of such methods to an access points list, then best first try out the desired (fixed) search statement(s) in the *Expert search system* in your Adlib application.

Then copy the search statement to the *Query* property of your new method, here in Designer.
(This method type is available from Adlib 6.3.0.)

- *Merge terms* - Add this method to a data source to allow merging of terms in it.

If you have an Adlib SQL or Adlib Oracle database which is not yet "XML" multilingual, but has separate records for each translation of a term or name, and you either want to start storing all translations of a term or name in one record (the XML multilingual way), or you want to apply preferred-term relations between the translations, then you'll want to do this as efficiently as possible. For this purpose, Adlib (from 6.5.0) offers the *Merge terms* functionality. In a running application you can find this option in the *Edit* menu.

First make your application and database multilingual. A typical field to apply this to would be the Thesaurus *Term* field. A *Merge terms* method should subsequently be added to the relevant data source (in our example to the Thesaurus) in the application. In the *Search key (from)* property of the method, enter the field from which you want to merge terms, in our example: `term`. (Also give the method a logical name.) If you wish, you can protect this functionality from being used by everyone by assigning the proper access rights to this method.

Note that it is very important that the Adlib database (a table in an SQL database) in which you want to implement this functionality, has feedback links to all other databases in which there exist references to records from this database: this is because you will be removing term records or apply preferred-term relations!

See the "Merge terms" chapter in the Adlib release notes 6.5.0 for a step-by-step approach to setting up this functionality and then applying it in a running application.

- *Named range search* - Use this method to create an access point to search on a date range by just entering a period name which implies the relevant date range: the date range is specified in a record defining the period.

For example, imagine you want an Adlib Museum user to be able to use an access point *Production period date range* to search (truncated) on period names like `17th century`. You could decide to register these period names as terms in the Thesaurus under the *PERIOD* domain.

To be able to specify the date range, you would have to create two new date fields in the data dictionary of the Thesaurus and add these fields as a *Date (early)* and *Date (late)* entry field on the

relevant screen (*thes.fmt*).

In the *Collect* database you can use the existing *production.period* field (tag *PT*) linking to the Thesaurus *term* field. On the *Relation fields* tab of this field definition, in the *Range start field* and *Range end field* properties, you'd have to enter the two new date fields. In the Adlib Museum application structure you would then add a *Named range search* method called `Production period date range` for instance. In its properties you would enter the already existing *Date (early)* en *Date (late)* fields from the *Collect (!)* database in the *Search key (from)* and *Search key (to)* properties respectively; Adlib will perform the actual date range search in these fields. In the *Names index* property you'd then enter the name of the linked field or tag to the period names in the Thesaurus, in our example *production.period*. In the *Domain name* property you would enter `PERIOD`.

After you've saved all changes and have restarted the application, the new access point from the example is available in the *Internal object catalogue*. Period names or the first few letters of such a name can be used to search on (after you've registered some period names of course). Adlib will search the Thesaurus and found search keys will be presented in the *Search wizard*. After the user has selected the desired key, the appropriate Thesaurus record will be looked up to retrieve its date range. The date range is then automatically used to perform a date range search in the production *Date (early)* and *Date (late)* fields in *Collect*.

- *Location change procedure* - This method is required in the data sources for which you want to switch the *Change object locations* functionality on: without this method, the *Change locations* button in the *Edit* menu of your Adlib application won't become active for marked records. This applies to all application versions. This functionality allows the user to change the current location of a marked set of objects. This is handy if a part of or all of the collection changes location: with the *Change locations* option you can register such a change in one batch procedure, instead of having to edit each record separately.

The method is present by default in some data sources in model application 4.2 and higher, when the data source has a *Location* and a *Location history* screen. To switch this functionality on in older applications, you'll have to add this method to the relevant data source(s) yourself. Aside from a name for the method, no properties have to be filled in.

Besides switching this functionality on, the method allows you to set access rights for it, if you don't want everyone to be able to use the *Change object locations* dialog of the procedure. *None* or *Read* access rights disable this functionality.

In model applications 4.2 and higher, running on adlwin.exe 6.5.0 or higher, the user gets to see a more advanced *Change object locations* dialog than present in older applications, fitting the extended and changed registering of locations in 4.2. You can run model applications older than 4.2 on adlwin.exe versions 6.5.0 and higher as well: the software will automatically recognize old and new applications to decide which *Change object locations* dialog must be presented to the user. (The software actually checks for the presence of tag 2A (*current_location*) in the collect database to decide whether the application version is 4.2 or higher.)

- *Print barcodes* - Use this method only if you want to protect single-click printing of labels to a label printer (as introduced in Adlib 7), via the Adlib access rights mechanism. By default, all Adlib users can print to labels (if label templates have been set up as described in the Adlib User Guide). If you don't want that, you'll have to add the *Print barcodes* method to the data sources which need some protection. Selected roles can be assigned limiting access rights on the *Access rights* tab of the method properties. With *Read* access rights, users can still print, with *None* they can't anymore.
- *Publish records to TheCollectionCloud* (available from Adlib Designer 7.1.0.30) - To enable the uploading of records to The Collection Cloud, no setup is required: by default you can upload records from within the *Objects*, *Archives* and the *Library catalogue* (representing the `collect` and `document` databases). However, by adding the *Publish records to TheCollectionCloud* method to the desired data sources in your application definition, you have the possibility to set up access rights for this functionality, to limit its use. Selected roles can be assigned limiting access rights on the *Access rights* tab of the method properties. If you do not set access rights, then in principle all users still have the possibility to upload and remove records (as far as access rights set on a higher level do not prevent that, of course). Further note that uploading cannot be separated from the possibility to delete records from The Collection Cloud: the *Full* and *Write* access rights allow for uploading and deleting, while *Read* and *None* access rights switch both functions off for the relevant users.

Menu texts

You must provide the name of the method (if it is an access point) as it will appear in the access points list (step 2) in the *Search wizard*, in as many languages as you wish to make available to users. It is useful to also provide menu texts for methods that are not access

points, as they will be used in the tree view in the *Application browser* of Designer.

Search key (from)

Enter or search for the tag or field name of the field in the current database, that you want to associate with this method, if it is an access point (usually *Term* and *Free text search* method types). There must have been defined an index for this field in the database setup. The key type of the relevant index will determine the default sorting of the search result.

If you create an access point for a field which is linked to an external database by means of a forward reference, you must enter the index tag for the forward reference field here. The Adlib user can search on the data in the current data source just as if there were no external link.

If you want to make an access point for a merged-in field, then enter or search for the tag or field name of the destination field in the current database. In this case, no index must be present for the destination tag in the current database; a term index on the source tag (from which the merged-in value is retrieved), does need to be present in the linked database. The source tag in the linked database cannot be a linked field itself.*

For methods of the *Date range search* or *NamedRangeMethod* type (for an explanation, see above), enter or search for the tag or field name of the (indexed) date field which contains the "early" date.

* If you want to create an access point for a merged-in tag which is a linked field itself in the linked database, you must base this access point on a fixed query. Just set the *Method type* option above to *Fixed query* and set the desired query in the *Fixed query* option below. An example of such a fixed query for the *collect* database would be `au = %data%*`. This would allow the user to use this access point to search on any author of documentation linked to an object record; author is a merged-in field here, from the *document* database, while in there it is a linked field to the *people* database.

Search key (to)

For methods of the *Date range search* or *NamedRangeMethod* type (for an explanation, see above), this property becomes active. Enter or search for the tag or field name of the (indexed) date field which contains the "late" date.

Names index

For methods of the *NamedRangeMethod* type (for an explanation, see above), this property becomes active. Enter or search for the tag or field name of the (non-indexed) data dictionary field which links to the (indexed) field (in another database) containing period names and their associated date ranges.

Domain name

If this method is an access point (usually *Term* and *Free text search* method types) and if searches with this access point only need to be done in a certain domain in the above specified *Search key (to)* property in the current data source, then enter here the name of the domain. Domains must of course be included in the index for this field. Type the domain name exactly as it is defined in the database setup in the *Value* column of the enumerative list for the domain field, because domain names are case-sensitive.

Truncation

For each access point you can set truncation differently. Choose from:

- *Default* - For text (term) indexes this means automatically truncating right. For free text (word) indexes, if the *Truncate free text searches by default* option for an application has been marked, setting the current option to *Default* means automatically truncating right, otherwise it means turning truncation off. In Adlib SQL databases (not in CBF databases), you can extend the default right truncation in a search with left and/or middle truncation by entering an asterisk to the left of the search key and/or somewhere in between the letters of the search key. For example: searching on *m*n could find a term like *dominant*.
- *Right* - This sets truncation to automatically apply on the right side of every search key. With for example the search key *put* you might find terms like *putrid* or *putting green*, but not *output* or *computer*.
This setting overrules the *Truncate free text searches by default* setting on application level, for the current method.
In Adlib SQL databases (not in CBF databases), you can extend the right truncation in a search with left and/or middle truncation by entering an asterisk to the left of the search key and/or somewhere in between the letters of the search key. For example: searching on *m*n could find a term like *dominant*.
- *None* - This turns automatic truncation off. Now an asterisk needs to be used to truncate. (So with *None* you search on the whole search key only, unless you use an asterisk to truncate manually.) In a CBF database search you can only use one

asterisk at a time, while in an Adlib SQL database search you can use multiple asterisks at a time to search left, middle and/or right truncated at the same time.

If in the CBF database(s) for which this method will be used, the possibility of left truncation has been set, the user can choose whether to place the asterisk on the left side of the search key (and truncate left), or place the asterisk on the right side of the search key (and truncate right). You can't search left and right truncated at the same time by using an asterisk on both sides of the key, though. (In a CBF database, you can only search left and right truncated at the same time in the Adlib *Expert Search language*, using the relational operator *contains*.)

In an Adlib SQL database, the possibilities of right, left and middle truncation are present by default, and you can use them all at once by placing asterisks anywhere in the search key.

This setting overrides the *Truncate free text searches by default* setting on application level, for the current method.

- *Left* - Set truncation to automatically apply on the left side of every search key. With for example the search key `put` you might find terms like *output* or *input*, but not *putrid* or *computer*. For this setting to work, the possibility of left truncation must have been set in the database(s) for which this method will be used, if these database are CBF database; for Adlib SQL databases no such setting is required. This setting overrides the *Truncate free text searches by default* setting on application level, for the current method.

In Adlib SQL databases (not in CBF databases), you can extend the left truncation in a search with right and/or middle truncation by entering an asterisk to the right of the search key and/or somewhere in between the letters of the search key. For example: searching on `m*n*` could find a term like *dominant*.

Fixed query

For a method of the *Fixed query type*, you have to fill in a search statement here. The syntax of such search statements is largely the same as for those in the *Expert search system*: the only difference is that from Adlib 6.5.0 you may include in this query the variable `%data%` and/or `%worddata%` (from 6.6.0) one or several times instead of one and the same literal value in the search statement. Either variable allows the user to enter one or more search keys after choosing the relevant fixed query "access point". An exception are pointer files: to request a pointer file by its number, use the fixed query `pointer %numericdata%`.

If you place an asterisk behind the variable, like `%data%*` or `%worddata%*`, it means that automatic truncation is implied in the search: the

user can enter a partial term, and Adlib will find all terms starting with that partial value. The only difference between the two variables is how truncation is applied to multiple search keys: `%data%*` will only consider the last entered search key truncated, while `%worddata%*` will consider all entered search keys truncated. It is recommended to use `%worddata%*` for searches with the equals operator (=) on word-indexed long text fields because multiple entered search keys may represent different words from a text, while you can use `%data%*` often probably best for searches with the equals operator on term indexed fields because multiple entered search keys may represent a single term or represent the first part of that single term. If you want to use the `contains` operator in a fixed query, you must not truncate the `%data%` or `%worddata%` variable because the `contains` operator already implies truncation.

Suppose you wish to offer a fixed query via the current method in the current data source, e.g. a new method named *Recent accessions* in the data source *Books* of a Library application, to request the accessions of the past 30 days. A proper query would then be: `acquisition_date from "today-30"`. This query has no `%data%` or `%worddata%` variables, so it will be executed immediately after the user chooses the access point.

Another example is a query with which you search for books which have been acquired in the past month and have been written by an author to be entered by the user after choosing this access point. This method could for instance be named *Accessions for author*. A proper query would be: `au = %data%* and du from "today-30"`. The asterisk indicates that any search value entry must always be considered truncated.

For putting together a search statement as a fixed query, there are some considerations to take into account:

- In the previous example, the *Search wizard* worked as expected, because truncation had already been specified in the search statement. If you wouldn't set it up like that, you'd have to tell the user in some way that on entry, partial names have to be truncated explicitly using an asterisk. And this is not obvious in the *Search wizard*, since truncation is usually applied implicitly there.
- One reason for the `%data%/worddata%` functionality is the wish to be able to search multiple fields on one and the same value simultaneously. Indeed, the `%data%` and/or `%worddata%` variable may appear more than once in the fixed query. An example of such a search statement in a Museum application could be: `OB contains %data% or TI = %worddata%* or BE = %worddata%*` With this method, you allow the user to search the *Object name*,

Title and *Description* fields simultaneously and automatically truncated. For fixed queries like this, you should ask yourself how the user will enter his or her search key(s). How should truncation be applied and how should multiple entered search keys be treated? A few guidelines can be given:

- Usually, you want to be able to search on a value which must occur in at least one of the specified fields. So combine the simple search statements in the fixed query mostly with OR, like in the Museum example.

- In term fields, like in *Object name* (OB), multiple words may occur. The second or one of the following words can only be found if you use the `contains` equation in the relevant simple (partial) search statement, and then behind `%data%` no asterisk is required because `contains`-searches are always truncated implicitly. In free text fields like *Title* and *Description* this is not required, so there you can use the `=` operator.

- If the user enters multiple search keys, separated by spaces, then for term indexed fields Adlib regards those to be one key together, while for word indexed fields Adlib regards those to be separate keys when compared with `=` and a single key when compared with `contains`.

- If the user enters multiple search keys, separated by spaces, all keys must occur in one and the same field. However, maybe you would like an OR relation to exist between the search keys, so that at least one of the entered search keys must occur, not necessarily all. There is no way to define this in the fixed query though. Therefore you'll have to tell the user in this case to apply a comma and space to separate multiple search keys (a comma equals OR).

- An attractive fixed query may be: `record contains %data%` With this method you offer a very simple "access point" with which the entire database will be searched (truncated) on an entered search key. The search may take a (very) long time though, if your databases are large.

Initial screen

When you use the *Search wizard* to search for records via an access point, and you open one of the found records in detailed display, then by default Adlib shows the first tab of such a record in detail first. But that is not always handy. If, via an access point, you search on a field which is not on that first tab, then you may first want look for the

tab that does hold the field, to confirm that the search result is correct.

That is why, from version 6.3.0, Adlib offers the possibility to have a record opened on a specific tab automatically, after searching via an access point, for instance the tab on which the relevant field can be found.

However, this has to be specified per access point first. Here, in the *Initial screen* option of the desired selected method, choose from a list of all screens linked to the current data source, to associate it with the current access point. In this context, this screen will be used to show to the user first upon display of a record which was found after searching on this access point.

Sorting adapl & Fields

In the *Sorting* option group, you can set the sort criteria that Adlib must use to determine the order of the records as presented on brief display screens, but you only need to set this option if the default sorting, as determined by the *Key type* of the index searched by this access point, is not what you want.

These sorting options are only editable if the current method is an access point. You can specify sort fields and additionally possibly a sort adapl. If you do not specify sort criteria, Adlib will sort the retrieved records ascending on the currently searched field, according to the key type of its index.

If you want to use an adapl to influence the sorting of records, you can enter the path to it here, or search for it on your system. A sort adapl doesn't actually sort anything though. You could write an adapl to fill some (temporary) tag with a compound value (put together from other fields in the currently processed record), and then sort on that temporary tag by setting it in the *Field* list. So the indicated fields determine on which fields the search result will be sorted, and an optional sort adapl can fill one or more of those fields before sorting is executed.

The sorting order in which fields are listed in the *Sorting Fields* property determines the order that Adlib uses when sorting the search result. You can of course add, change or remove sort fields. If you add or change a sort field, you'll have to set four properties for it:

- **Field**
Enter or search for the field tag or name on which to sort. (Tags are case-sensitive.)
- **Sort all occurrences**
Here you indicate whether you want to sort on the first occurrence only (*False*) or on all occurrences (*True*). If you sort on all occurrences of this field, a record appears as much times in the

search result as this field has occurrences in that record.

- **Sort key type**

Select a sort key type, preferably according to the field type of this sort field:

Text - Sort alphabetically according to the host character set.

Numeric - Sort in numeric order.

Date - Sort in date order.

- **Sort order**

You can display the records in *Ascending* (a-z) or *Descending* (z-a) order.

Help key

Here you can enter the key to the Help text which will be displayed if the user presses **F1** (Help) while the selection bar is on the current access point in the access points list (step 2 in the *Search wizard*) in a running Adlib application.

To some methods, like *Create new records*, *Expert search*, or *Query by form*, this option is not applicable, and therefore greyed out.

3.12.5.3.2 Access Rights

On the *Access rights* tab, which is present when you have selected a new or existing method in a data source in an application in the *Application browser*, you can restrict access to this method, dependent on the user (login name in Windows) and its assigned role.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this method. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. Access rights for a role associated with a method can never supersede access rights for the same role associated with a dataset or database. If a role is not linked to this method, then each user linked to that role has full access by default. If no role is linked to the method, every user linked to a role has full access. A user without a role always has full access. Users are assigned to roles in the application setup.

Some access rights make nonsensical combinations with certain methods, like a *Create new records* method with read-access. But it shouldn't cause errors. The result is usually that different access rights have the same effect. For *Term search* and *Free text search* methods for instance, there are basically just two choices: you want

to allow searching on this access point, in which case *Read*, *Write* and *Full* are equally applicable and have the same effect, or you don't want to allow searching. In which case you choose the access rights *None* and the method won't even appear in the access points list in step 2 of the *Search wizard*. And for the *Create new records* method the access rights *None* and *Read* have the effect that the user is not allowed to create new records (the button for it won't be present), while *Write* and *Full* do allow creating records. An overview of all combinations can be found here. In general, you'll be able to use common sense to figure out what effect certain access rights will have on a method.

Do note that applying limiting access rights on method level may not be as safe as applying them on database level. For instance, excluding users from being able to create new record in a data source, still allows them to create new linked records in it from another data source to which they do have write access! If you want to make it impossible for certain users to create records, you are better off setting these access rights in the database setup.

See also

Security in Adlib

3.12.5.3.3 Screen references: properties

On the *Screen name properties* tab, which is present when you have selected a new or existing screen reference for a method in a data source in an application in the *Application browser*, you choose an (existing) screen file for this reference. A screen reference for a method has priority over screen references specified for the data source. If you don't specify a screen reference for a method, by default the screen references defined for the current data source will be used.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll only find the following setting:

Relative path

Select the desired screen file to associate with this method. Preferably use a relative path, for example: `../screens/qbfdoc` for a search screen associated with a *Query by form* method in a *Books* data source.

Typically, in existing Adlib applications only for this method, one search screen reference has been defined. All other screens

references are specified for the data source, to apply to all methods therein.

Note that there are three types of screen references: *List screen*, *Detail screen*, *Search screen*. You can choose the type when you create a new screen reference for a method in the tree view in the *Application browser*. Make sure you select a screen file of the same type, when you set this screen reference.

In very old applications you may also find for every access point one search screen reference that may even point to a screen file that is no longer present. These search screen references for access points are a remnant of the past and were used as the search screen after the user chose an access point; if no search screen reference for an access point had been defined, the default search screen reference for the data source was used. For Adlib Windows applications this setting has become obsolete. Now the software will automatically generate the appropriate search screen through the *Search wizard*, depending on what method was chosen by the user. So you don't need to create any search screen references for access points, but if they already exist, you can leave them as they are too (including any faulty links to non-existent screen files).

3.12.5.4 Screen references

3.12.5.4.1 Screen name properties

On the *Screen name properties* tab, which is present when you have selected a new or existing screen reference for a data source in an application in the *Application browser*, you choose an (existing) screen file for this reference.

Note that screen references for methods have priority over screen references specified for the data source. Only if you don't specify a screen reference for a method, by default the screen references defined for the current data source will be used.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll only find the following setting:

Relative path

Select the desired screen file to associate with this data source. Preferably use a relative path, for example: `../screens/br_catal` for a list screen associated with a *Books* data source, or for example: `booka` for a detail screen in the same data source - this screen file is located in the current folder.

Typically, in existing Adlib applications most screens references are

specified for the data source, not for specific methods (except for the *Query by form* method). Usually, a data source *Screens* list contains the following:

- one list screen reference for the brief display that Adlib must use to display matching records after a search in this data source. Note that there are three types of screen references: *List screen*, *Detail screen*, *Search screen*. You can choose the type when you create a new screen reference for a method in the tree view in the *Application browser*. Make sure you select a screen file of the same type, when you set this screen reference;
- a number of detail screen references that together display the data from one record in this data source. Adlib will display the detail screens to the user in the order of the current *Screens* list in the tree view;
- possibly one search screen reference that may even point to a screen file that is no longer present. This is a remnant of the past and was used as the default search screen after the user chose an access point: for Adlib Windows applications this setting has become obsolete. Now the software will automatically generate the appropriate search screen through the *Search wizard*, depending on what method was chosen by the user. So you don't need to create any search screen reference for a data source, but if it already exists, you can leave it as it is too (including any faulty link to a non-existent screen file).

3.12.5.5 Output jobs

3.12.5.5.1 Output job properties

On the *Output job properties* tab, which is present when you have selected a new or existing output job for a data source in an application in the *Application browser*, you specify the output format (adapl and/or Word templates, or XSLT stylesheets for printing) that must be available from within the current data source. Adlib displays the output jobs that you specify here, in the *Select output format* list in the *Print wizard* dialog of an application, if the user has chosen to *Create a report with a predefined output format*.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Adapl

If you want to use an adapl for this output format, then select the desired ADAPL procedure on your system. You can also type the path and name of the adapl yourself. Do not enter an extension.

With an adapl you can process any data and print the results as plain text, to a file or to a Word template as you see fit, or leave the result in memory to be exported to a Word template by Adlib.

If you also want to use a Word template, then this adapl may not contain any `PRINT` or `OUTPUT` statements. In this case, the adapl is often used to pre-process data. The Word template can be called from within the adapl or by Adlib after the adapl has finished, by using the template set in the property below. Which option to use, depends on what you want to achieve. The second option is the only one if you want to print to a labels template, while the first option may be the only option if you want to collect data from more than one record before printing to the template. If you call the template from the adapl, you control the moment and arguments of the call, while letting Adlib do the call just transports all relevant data to the template after each record.

If you use a Word template set in the property below in combination with this adapl, you may want to transport some collected data from the adapl to the template. To the template, tags from the currently processed record are available by default and referenced in the template like so: `<<tag>>`. But if you collect and process data from several fields to be transported to the template as one field, you cannot store this data in one of the existing tags nor in a variable. You solve this by creating a dummy FACS database with tags that do not exist in the data dictionary, for instance:

```
fdstart DUMMY './data+document'          /* dummy database
Q1
Q2
Q3
fdend

open DUMMY
clear DUMMY
```

The referenced database (in this case *document*) is irrelevant, as long as the tags do not exist in there. Use the tags to temporarily store any compound information in; you don't need to write anything back to the database. After the adapl ends and the template is processed, the dummy tags are still present in memory and can therefore be transported to the template. Just use references in the template as follows: `<<DUMMY+Q1>>` to indicate that the tags are present in the DUMMY FACS database (at the moment at least), not in the local record.

Note that you shouldn't use `<tag>=null` assignments in the `adapl` if the tag is part of a dummy FACS database: this causes errors when printing to a Word template. Instead, use `<tag>=''` to assign an empty value.

Template path

For this property you may select a Word template (a `.dot` file from Word 97 or higher) on your system to set up as an output format. The way you design the template (see the Adlib User Guide) determines if every record is printed on a separate page or not, and what tags and how many occurrences will be printed. A label template also requires a special design.

You can use an `adapl` to pre-process data, if you wish, before the template is processed. Such an `adapl` can be set in the property above.

A wildcard `*` (asterisk) can be used instead of the path to a template. When the user selects such an output format in the *Print wizard*, the *Template* window opens. From this window, the user can select any template on the computer or network to print to. After selecting a template, it will be filled with the data from the marked records. Use the wildcard in combination with an `adapl`, if you want to pre-process data from marked records before the data is sent to the Word template. Upon printing, the user will have to select the `adapl` as a predefined output format first, then go through the Windows default *Print* window, after which he or she will be allowed to pick any template from the file system. The tags in the template will be filled with record data after all or part of it has been processed by the `adapl`.

Template type

The *Template type* indicates the type of template (if you use one): *Label* is a Word template for making labels and *Normal document* is for a normal document template.

Stylesheets XML type

If you are using an XSLT stylesheet as an output format (in which case the stylesheet(s) have been specified in the option below), you must indicate here which XML type is expected by the stylesheet(s): *Grouped* or *Unstructured*. By default, Adlib (`adlwin.exe`) generates unstructured XML and any XSLT stylesheet output formats created prior to Adlib 7.1 will always be based on unstructured XML. From Adlib 7.1, the new *XML type* option will also default to *Unstructured* for existing output formats; in new output formats though, it will default

to *Grouped*. Only from Adlib 7.1, you can choose whether to base your stylesheet on grouped or unstructured XML. Click here for more information about the differences between the two formats.

Stylesheets Path

This option allows you to set one or more XSLT stylesheets to be used as an output format. XSLT stylesheets are UTF-8 encoded text files with the extension *.xsl* or *.xslt*. Only XSLT 1.0 is supported, not XSLT 2.0.

All entered stylesheets for an output format are executed in sequence, from top to bottom, when printing. This allows you to spread complicated transformations over different stylesheets. XSLT stylesheets cannot be used in combination with an adapl or Word template.

Job title

Enter a name for this output format in one or as many languages as you wish to make available to users. These names will be used in the *Print wizard* dialog.

In adapls, titles may already have been specified; these have priority over any titles specified on this tab. If no titles have been defined in the output adapl, the titles specified on this properties tab will be used.

Job description

To elucidate this output format you may provide short descriptions. These will not be visible to the user.

3.12.5.5.2 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing output job for a data source in an application in the *Application browser*, you can restrict access to this output job, dependent on the user (login name in Windows) and its assigned role.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this output job. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. Still, *Write* and

Full access have the same effect here: the output job is visible to the user and can be printed with. *None* and *Read* access both have the effect that this output job is not visible to the concerning user. Access rights for a role associated with an output job can never supersede access rights for the same role associated with a dataset or database. If a role is not linked to this output job, then each user linked to that role has full access by default. If no role is linked to the output job, every user linked to a role has full access. A user without a role always has full access. Users are assigned to roles in the application setup.

See also

Security in Adlib

3.12.5.6 Export formats

3.12.5.6.1 Export formats

On the *Export format properties* tab, which is present when you have selected a new or existing export format for a data source in an application in the *Application browser*, you specify the export format (adapl and/or Word templates) that must be available from within the current data source. Adlib displays the export format that you specify here, in the *Export* wizard dialog in an application.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Adapl

If you want to use an adapl to define the layout of this export format, then select the desired ADAPL procedure on your system. You can also type the path and name of the adapl yourself. Do not enter an extension.

If you also want to use a Word template, then this adapl may not contain any `PRINT` or `OUTPUT` statements.

Template path

For this property you may select a Word template (a *.dot* file) on your system to set up as an export format. Also the wildcard * (asterisk) can be used instead of the path to a template. When the user selects such an export format in the *Export* wizard, the *Template* window opens. From this window, the user can select any template on the computer or network to export to.

Template type

The *Template type* indicates the type of template (if you use one): *Label* is a Word template for making labels and *Normal document* is for a normal document template.

Stylesheets XML type

If you are using an XSLT stylesheet as an export format (in which case the stylesheet(s) have been specified in the option below), you must indicate here which XML type is expected by the stylesheet(s): *Grouped* or *Unstructured*. By default, Adlib (*adlwin.exe*) generates unstructured XML and any XSLT stylesheet export formats created prior to Adlib 7.1 will always be based on unstructured XML. From Adlib 7.1, the new *XML type* option will also default to *Unstructured* for existing output formats; in new output formats though, it will default to *Grouped*. Only from Adlib 7.1, you can choose whether to base your stylesheet on grouped or unstructured XML. [Click here](#) for more information about the differences between the two formats.

Stylesheets Path

This option allows you to set one or more XSLT stylesheets to be used as an export format. XSLT stylesheets are UTF-8 encoded text files with the extension *.xsl* or *.xslt*.

All entered stylesheets for an export format are executed in sequence, from top to bottom, when exporting. This allows you to spread complicated transformations over different stylesheets. XSLT stylesheets cannot be used in combination with an adapl or Word template.

Job title

Enter a name for this export format in one or as many languages as you wish to make available to users. These names will be used in the *Export wizard* dialog.

In adapls, titles may already have been specified; these have priority over any titles specified on this tab. If no titles have been defined in the export adapl, the titles specified on this properties tab will be used.

Job description

To elucidate this export format you may provide short descriptions. These will not be visible to the user.

See also

Export format access rights

3.12.5.6.2 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing export format for a data source in an application in the *Application browser*, you can restrict access to this export format, dependent on the user (login name in Windows) and its assigned role.

[Click here](#) for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this Export format. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. Access rights for a role associated with an export format can never supersede access rights for the same role associated with a dataset or database. If a role is not linked to this export format, then each user linked to that role has full access by default. If no role is linked to the export format, every user linked to a role has full access. A user without a role always has full access. Users are assigned to roles in the application setup.

See also

[Security in Adlib](#)

3.12.5.7 Friendly databases

3.12.5.7.1 Friendly databases

On the *Friendly database properties* tab, which is present when you have selected a new or existing friendly database for a data source in an application in the *Application browser*, you refer to compatible databases for the current data source.

[Click here](#) for information on how to edit properties in general. And [click here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Menu texts

Specify the title of this compatible database in as many languages as

you want to make available to users. Consider including information about whether the original record will be kept or will be deleted after deriving.

The titles of one or more friendly databases specified for the current data source, will be shown in the submenu that opens when the user clicks *Start > Derive* in an Adlib application.

Folder

This property contains the full or relative path to the database you want to link to, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the relative path to that folder is automatically entered here. Usually, you keep your databases in one folder, and the relative path you'll often encounter here is: `../data`.

Instead of a path to a local database folder, you may enter a base query URL to an HTTP handler (gateway) for an external source, if you want to be able to derive records from that external source. See the Help topic "Approaching external sources as friendly databases" for more information.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you want to link to. Do not enter the extension of the file. Examples of database names are `document`, `copies`, and `collect`. (This option does not apply to external sources accessed via a gateway.)

Important: when you work in a copy of your live application, then make sure you search the right folder (in the copy) for the proper file: otherwise the relative path will be incorrect when you place back this copy as your live application later on.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) Typically, you select a dataset if for this link you only want to retrieve data from that specific dataset.

There is a difference between linked field processing for records derived from a dataset in the current database and records derived from another database (or dataset in another database), namely that in the first case for reference linked fields the link references (record numbers of linked authority file records) and only these references are copied to the new record. This is the optimal solution because the new record will contain the same reference as the original, and we can be sure the reference can be resolved and that merged fields can

be retrieved because both records exist in the same database and will use the same authority files. But in the second case we cannot be sure whether the other database uses the same authority files as the current. Therefore we can't just copy the link reference to the new record: after all, it might refer to a completely different record. So, in this case only the linked values and merged field values themselves are copied to the new record. The existing link reference will not be copied along, and neither will a new reference link be created directly and automatically for the value in the new record; the latter being important for authority files linked to the current database, which have non-unique indexes. An automatic reverse resolving of a non-unique value would pick the first authority record number for that value, while that may be the wrong one. On saving the new record, all linked fields will automatically be reverse resolved to the authority files linked to the current database, but you'll have to manually select the proper authority records for terms which are not unique, before the record can be saved.

Search screen

Select the screen to be used as a *Query by form* for searching in the friendly database. Preferably, use relative paths like `../screens/qbdoc`, and name the screen without extension. A search screen is mandatory.

A *Query by form* is necessary because the link to the friendly database only retrieves record numbers as the key to the "friendly" records; in the *Query by form* the user can usually enter several criteria to search for the needed record number. Therefore, all the fields on a *Query by form* must be present in the friendly database. If the search performed from such a *Query by form* yields more than one result, these records will be listed in a *Linked record search screen*: this is either a standard link screen for the linked record number field, or the *Search result screen* specified in the option below. The user makes the final choice from this list.

A search screen needed to set up one of the available external sources as a friendly database, can be obtained from the Adlib Helpdesk.

Search result screen

Select the link screen which Adlib must use to display a list of possible values from specific fields of the "friendly" records, in the *Linked record search screen*, if the search found more than one result. If you specify no link screen in this option, Adlib will display the standard *Linked record search screen* automatically when the search yields more than one record. This standard *Linked record search*

screen in this case only contains one column of record numbers (which is not very informative).

A link screen contains only two lines of screen elements. The top line consists of labels for the fields below them in the second line. Adlib doesn't display this screen like other normal record screens in an application: the link screen is converted to the *Linked record search screen*: with the labels on the link screen you specify the column headers, and with the fields the contents of the columns in the *Linked record search screen*.

A search result screen needed to set up one of the available external sources as a friendly database, can be obtained from the Adlib Helpdesk.

Zoom screen

Select the screen in which Adlib must display (and only for display, not to edit) details of a "friendly" record, when the user clicks the *Show* button for a selected record in the list in the *Linked record search screen* that may result from a search in the friendly database.

You can base your zoom screen on an existing detail display screen: often the first detail display screen to display a record in the friendly database is good enough. But in the zoom screen the *Read-only* property for the screen file must be set, otherwise the user can make changes in the friendly record through the zoom screen, which is not desirable.

There is no default zoom screen, so the user cannot zoom to a friendly record without a zoom screen specified for it (the *Show* button will be greyed out).

A zoom screen needed to set up one of the available external sources as a friendly database, can be obtained from the Adlib Helpdesk.

Remove original record after retrieval

If you do not want to keep the (original) record that a user derives from the friendly database, you mark this checkbox.

If you leave the checkbox unmarked, deriving a record will copy it, leaving the original in place.

(This option does not apply to external sources accessed via a gateway.)

After deriv procedure (ignore "After")

It is possible, if desired, to have an *adapl* executed just before *and* just after a record has been derived (still before storage). So the *adapl* you set here is executed twice. In your ADAPL code, use the `&1`

execution codes 24 and 26 to determine if the `adapl` is executed before or after `deriv`. Also see the reserved variables `&0`, `&0[2]` and `&0[3]`.

3.12.5.7.2 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing friendly database for a data source in an application in the *Application browser*, you can restrict access to deriving records from this friendly database, dependent on the user (login name in Windows) and its assigned role.

[Click here](#) for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this friendly database. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. Still, *Read*, *Write* and *Full* access have the same effect here: the user can derive records from this friendly database. Access rights for a role associated with a friendly database in the application setup can never supersede access rights for the same role associated with a dataset or database in the database setup. If a role is not linked to this friendly database, then each user linked to that role has full access by default. If no role is linked to the friendly database, every user linked to a role has full access. A user without a role always has full access. Users are assigned to roles in the application setup.

See also

[Security in Adlib](#)

3.12.6 Users

3.12.6.1 User properties

On the *User properties* tab, which is present when you have selected a new or existing user in an application in the *Application browser*, you link individual users of this application to new or existing roles.

[Click here](#) for information on how to edit properties in general. And [click here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following

settings:

User name

This property must contain the name of a user of this application. This name must be exactly the same (case-sensitive) as this user's login name in Windows.

Password

You only have to fill in a password here if you use the user authentication method called "Adlib.PBK" (in which case the user must always log in to Adlib with his or her user name and this password. The password doesn't need to be the same as this user's password in Windows.

Role

Each user must be assigned a role (a user may have only one role), for example: `student`, `management`, `system manager`, `employee`. You can enter a new role, or pick one from the drop-down list that displays the roles already present in this application.

See also

Security in Adlib

Users and roles

3.12.7 FACS

3.12.7.1 Database alias (FACS)

<This functionality is deprecated.>

On the *Database alias* tab, which is present when you have selected a new or existing database alias in an application in the *Application browser*, you can set up an alias to a database or dataset, to be used in adapls that are executed in the current application. However, this functionality is no longer in use in current model applications. The preferred method is to define these database aliases in the adapls themselves. If a FACS name is both defined with Adlib Designer *and* in the adapl, the definition in the ADAPL procedure takes precedence.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Name

Enter a name to be used to approach the following database or dataset under FACS. A FACS name can be up to 32 characters long, and may not contain any spaces.

Folder

This property contains the full or relative path to the database you want to link to, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the relative path to that folder is automatically entered here. Usually, you keep your databases in one folder, and the relative path you'll often encounter here is: `../data`.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you want to link to. Do not enter the extension of the file. Examples of database names are `document`, `copies`, and `collect`.

Important: when you work in a copy of your live application, then make sure you search the right folder (in the copy) for the proper file: otherwise the relative path will be incorrect when you place back this copy as your live application later on.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) Typically, you select a dataset if for this alias you only want to retrieve data from that specific dataset. Selecting a dataset (if available) is mandatory if you want the adapl to be able to write records to it.

See also

ADAPL programming: FACS

3.13 Properties of Adloan application objects

§

3.13.1 Adloan general properties

On the *Adloan properties* tab, which is present when you have selected a new or existing Adloan Circulation application in the *Application browser*, you determine the global settings for Adloan Circulation. The settings apply to all library branches.

Click [here](#) for information on how to edit properties in general. And click [here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Process fines

Mark this checkbox to turn fines processing on.

Leave this option deselected if your library doesn't fine borrowers for overdue copies.

Use library opening hours table for calculating fines

Mark this checkbox if you want the table with branch opening and closing times to be used in the calculation of fines. This means that for overdue materials no fine will be calculated for the overdue days on which the branch was closed.

Leave this option unmarked if you do not want the closing times to be taken into account, so that fines will be calculated for all overdue days, even if the branch was closed on one or several of those days.

Use library opening hours table for calculating return date and time

Mark this checkbox if you want the table with branch opening and closing times to be used in the calculation of the due date and time. This means that although the return date is calculated by adding the maximum loan period for the relevant borrower category, this date will be moved up to the first following day on which the branch is open, if the initially calculated date happens to be a day on which the library is closed.

Leave this option unmarked if you do not want the closing times to be taken into account. This means that the return date may also be a day on which the branch is closed.

Check copy status of all copies during transactions

Mark this checkbox if you want all copies, reservations and fines for a certain borrower to be checked at every transaction.

Leave this option unmarked if you do not want this check to take place. Without the status check you can work faster and get fewer messages during transactions, but you will get less information too.

Print issue slips

This option is no longer in use in Adloan Circulation for Windows.

Payments per location

This option allows for the possibility to have payment information during issuing, returning and reserving displayed only if that information pertains to the current site, even in the *Payments for borrower* window. So when the option has been set, loan fees, fines and subscription fees are only visible to co-workers of the location which must collect the payment. The home site of a copy is leading for the loan fees and fines while the site to which the borrower has been registered is leading for the subscription fee. By default, the option is switched off so that all payments remain visible to all sites.

Rounding method

Here, you can determine which rounding method must be applied to hours, when the time at which copies are to be returned needs to be calculated. There are four settings to choose from:

Option	Meaning
<i>Closest</i>	Rounding to the closest hour: 1-29 minutes will be rounded downwards and 30-59 minutes upwards.
<i>Up</i>	Always round upwards.
<i>Down</i>	Always round downwards.
<i>None</i>	Do not round off.

Loan categories

Loan categories indicate material types. Each item which can be issued must be assigned a loan category. Together with the borrower category, the loan category determines what loan limits, reservation limits and fine tables apply.

If a loan category which has not been defined for the branch occurs in a transaction, Adloan Circulation will use the *Default loan category* for the branch, as specified per library branch. If that doesn't exist either, the standard loan category will be used, as specified under *Default values*.

Loan categories must be listed here, as well as defined in Adlib Loans Management where each loan category is stored as a Thesaurus record. It is recommended that for this option you list all loan categories which you use in Adlib Loans Management, but there is no principle necessity for a univocal relation: so here you may word your

categories differently or just create one `normal` loan category. Do note that the categories listed here are the only ones presented in the *Loan category* drop-down list on the *Reservations* tab in Adloan Circulation.

3.13.2 Adloan data source properties

On the *Catalogue*, *Copies*, *Borrower status*, *Borrower category*, *Borrower*, *Fines*, *Copy reservations*, *Title reservations*, *Loan fees* and *Statistics* tabs*, which are present when you have selected a new or existing Adloan Circulation application in the *Application browser*, you specify the names and locations of databases and datasets, and the relevant tags in them, which are used by Adloan for handling and registering transactions.

If your `\data` folder has been installed in the standard directory (as a subfolder of an Adlib folder, next to the subfolders for the executables and possible other Adlib modules and applications), and in your Adlib databases you have never altered the names of datasets or tags, then the default settings on these tabs do not need changing. Only if the `\data` folder has been placed somewhere else, or when you changed names of files or tags, you should use this option to set up the Loans module with these changes.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

File path

Enter the absolute or relative path (the latter is preferred) to your `\data` folder, followed by a "+", then the database file name, and when you want to point to a dataset in there then a ">", and finally followed by the dataset name. The *Catalogue* path name, for example, is: `../data+document>fullcatalogue`

If you use a database without datasets, you can leave out the greater-than symbol and the dataset name.

Storage adapl

If, for loan transactions in the current database, you would like to use a different storage adapl than the standard storage adapl specified for this database in the data dictionary, then you may enter it here. This option is rarely used.

Example: if you wanted to write issuing transactions to a special

history file, you could write an adapl to do this (before or after) storage of transactions in the *Copies* database. The adapl could filter transactions using the reserved variable &5. You would link the adapl in the current property of the *Copies* data source of the Adloan Circulation pbk.

Fields

For each database field that is used by the Loans module, the tag must be specified here. In the standard installation of the system, fixed tag names are used. Only change these default tag names if absolutely necessary and you know what you're doing. Default tags are necessary for dealing with possible malfunctioning quickly and effectively.

In applications older than version 4.2, an *Image* field setting may be missing on the *Borrower* properties tab: this is because the possibility to set an *Image* field only became available in Adlib Designer 6.5.0. It allows Adlib Circulation to retrieve an image (e.g. a passport photo) which you linked to a borrower in Adlib Loans management, and to show it on every tab in Adloan Circulation when the user selects the relevant borrower.

To implement this functionality, you must add the *Image* field from the Borrower database, to the *Image* field property on the *Borrower* tab. By the way, Adloan requires the *adlibimg.dll* for this functionality. This dll should be located in the same folder as your Adlib executables (and is probably already present there).

* By default the following data sources are present (although invisible to the user) in an Adloan application:

Data source	Contents
<i>Catalogue</i>	The central loans database, with a record for each item (title or object) that can be issued. This is where the Loans module registers loan transactions for all items in circulation.
<i>Copies</i>	Contains details on each copy.
<i>Borrower status</i>	Contains information about the number of borrower transactions, and determines via the borrower category which limits and fine tables should apply to the borrower.
<i>Borrower category</i>	Contains the defined borrower categories. A borrower category determines which limits apply in total and per loan category (material

	type).
<i>Borrower</i>	Contains the borrower subscription details. It links the borrower to a certain borrower number and contains the term of validity of the subscription.
<i>Fines</i>	Contains information about fines and payments for each borrower for whom there is an outstanding fine.
<i>Copy reservations</i>	Contains reservation details per copy.
<i>Title reservations</i>	Contains reservation details per title.
<i>Loan fees</i>	Contains information on amounts owed and payments for each borrower for whom there is an outstanding loan fee.
<i>Statistics</i>	Contains counters that are logged per unique combination of transaction type - borrower category - loan category.

3.13.3 Default values

3.13.3.1 Default values

On the *Default values* tab, which is present underneath the node with the same name when you have selected a new or existing Adloan Circulation application in the *Application browser*, you specify a few default values. The settings apply to all library branches.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Messages file

The *Messages file* option is not used by the Loans module under Windows. (You may leave the value *MESSAGES.FIL* as it is, even though the file doesn't exist anymore: the option is ignored.)

Default loan category

The default loan category will be used if specific loan categories have not been provided for a specific location (library branch). Preferably, set this to "normal" (or the translation in your own language).

Default fine table

Default fine tables will be used if specific fine tables have not been provided for a location (library branch).

3.13.3.2 Screens

On the *Screens* tab, which is present underneath the *Default values* node and for a selected library branch when you are editing a new or existing Adloan Circulation application in the *Application browser*, you can specify default or branch-specific screens.

Click [here](#) for information on how to edit properties in general. And click [here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Screens

The *Screens* option is not used by the Loans module under Windows. You can leave any preset screens as they are.

3.13.4 Library branches

3.13.4.1 Library branch

On the *Library* tab, which is present when you have selected a new or existing library branch for an Adloan Circulation application in the *Application browser*, you specify settings and definitions for the current library branch a.k.a. location only. For locations for which these options are not specified, the *Default values* apply. Note that for the Loans module, every library location is a branch, even if there is only one location. In addition, all branches are seen as equal - therefore there is no separate main location.

Click [here](#) for information on how to edit properties in general. And click [here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Name

Enter a name with a maximum of 32 characters. In library location

names, a distinction is made between capitals and lower case letters. This means that `Main` and `MAIN` would be two different locations.

Messages file

The *Messages file* option is not used by the Loans module under Windows. (You may leave the value `MESSAGES.FIL` as it is, even though the file doesn't exist anymore: the option is ignored.)

Default loan category

Enter the name of the default loan category for this location. If no category is provided here, Adloan Circulation will use the loans category specified as the standard category under *Default values*.

Data folder

For this setting you can provide the path to a `\data` folder for the current library location, if that is different than the default `\data` folder. Normally, you don't have to fill in anything here.

Opening hours

Here you can specify per weekday at what time the branch opens and closes, and until what time items may be returned. Set both the *Opening time* and *Closing time* to `00:00`, to indicate that the branch is closed all day. By the way, after closing times or on a day on which the branch is closed, you may still use this Adloan Circulation application and carry out transactions.

Exceptions

Here, you may specify on which dates different opening, closing and/or return times apply, for example on public holidays. The date to be entered can be formatted in different ways, but you have to provide a year in four digits, so preferably use the format `dd/mm/yyyy`, for instance `31/12/2008`.

Closed periods

Here you may indicate during which periods the library will be closed. The dates themselves are included in the closed period. The dates to be entered can be formatted in different ways, but you have to provide a year in four digits, so preferably use the format `dd/mm/yyyy`, for instance `25/12/2008` up to and including `01/01/2008`.

3.13.4.2 Screens

On the *Screens* tab, which is present underneath the *Default values* node and for a selected library branch when you are editing a new or existing Adloan Circulation application in the *Application browser*, you can specify default or branch-specific screens.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Screens

The *Screens* option is not used by the Loans module under Windows. You can leave any preset screens as they are.

3.13.5 Users

3.13.5.1 User properties

On the *User properties* tab, which is present when you have selected a new or existing user for an Adloan Circulation application in the *Application browser*, you specify login information for every user of Adloan: every co-worker that is allowed to carry out loan transactions, has to be registered as a user, by means of his or her name and a password.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

User name

This must be the name with which the user will be logging on to the Adlib server. Capitals and lower case letters are treated differently. In a normal or demo installation there are two users present by default: *user1* and *gebruiker1*. It's your choice whether to leave these default user definitions present or not.

Password

Choose a unique password belonging to this username. Capitals and lower case letters are treated differently. With the standard or demo installation *user1* and *gebruiker1* have the password `pw1`.

Show issue tab

With this option you decide if the tab for issuing materials to borrowers is present when the current user has logged onto Adloan Circulation. Normally, for all issue desk employees this option should be marked. For a self-service user, this option would typically have to be marked.

Show discharge tab

With this option you decide if the tab for returning materials from borrowers is present when the current user has logged onto Adloan Circulation. Normally, for all issue desk employees this option should be marked. For a self-service user, this option can be unmarked.

Show reservations tab

With this option you decide if the tab for making reservations is present when the current user has logged onto Adloan Circulation. Normally, for all issue desk employees this option should be marked. For a self-service user, this option can be unmarked.

3.13.6 Fine tables

3.13.6.1 Fine table

On the *User properties* tab, which is present when you have selected a new or existing user for an Adloan Circulation application in the *Application browser*, you specify a fine table that must be used for this location, if the standard tables are not applicable. If a transaction requires a fine table that has not been defined for the location, then Adloan Circulation will use the standard fine tables, as specified under *Default values*.

Click [here](#) for information on how to edit properties in general. And click [here](#) to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Table name

Enter a unique name for this table of fines.

Day

Enter the number of "overdue" time units (days or hours) after which

the fine should be applied. Whether days or hours are calculated, cannot be set here: you'll have to do that in the running Adlib Loans Management application, per loan category - see the Loans module guide for more information.

The table allows you to provide multiple *Day-Amount* definitions.

Amount

Enter the fine amount (usually in cents) that must be calculated after the given period. You may only use whole numbers. The monetary unit to be used is implicitly determined by conventional practice of the library, and is not set in the system. You could also agree to work with penalty points, for example.

3.13.7 Self-service setup

Adloan Circulation 6.6.0 can be set up to allow borrowers to do their own issuing of materials, whilst the possibility to make reservations or to return materials can be disabled. With this setup, the user interface will automatically be limited to options and information to which borrowers are allowed access. With this functionality you can offer your co-workers in, for example, a company archive or internal library, an efficient way to do the registering and borrowing of materials themselves.

This method of self-service for borrowers is mostly suited for use with a trusted borrower group, because this setup is not secured in any way. (For a, via SIP2, secured self-service system for issuing materials, the Adlib software must be used in combination with special self-service desks.)

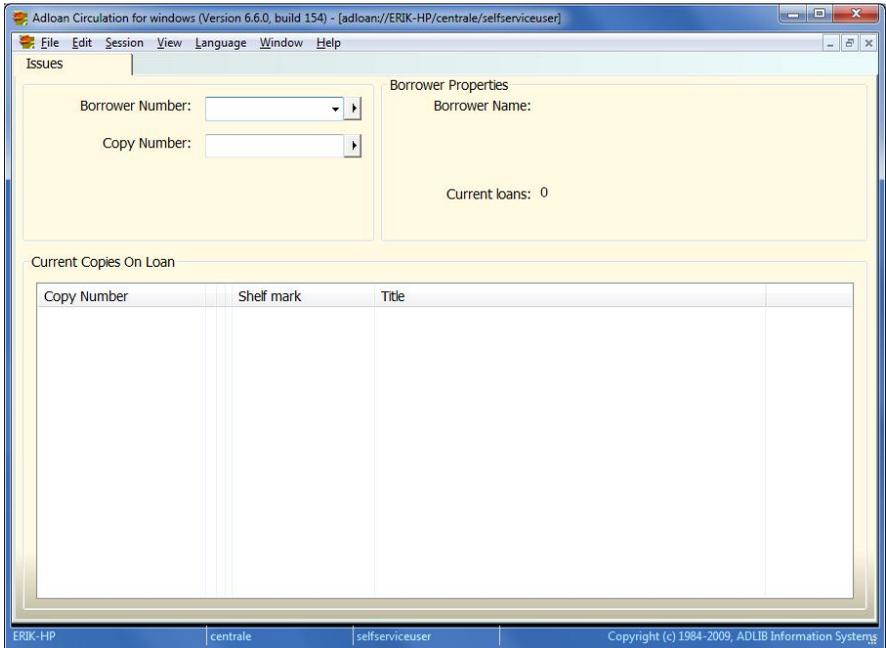
You can adapt your current Adloan Circulation application in such a way that all functionality remains the same for existing users (your issue desk co-workers), while for one new "user" (all borrowers who will be using the self-service later on) the new settings will apply. You make the required adjustments as follows:

Changing the application

1. In the Adloan application structure, create a new user under *Users*. The name can be chosen freely, but in this example we'll name this user "selfserviceuser".
2. Assign a password and write it down somewhere.
3. Subsequently only mark the options for the tabs which must be presented to the self-service users: *Show issue tab*, *Show*

discharge tab (returning) and/or *Show reservations tab*.

Only if you just mark the *Show issue tab* option, Adlib will also automatically limit the amount of information displayed on that screen tab, and the available functionality as well, so that under the *Borrower properties* only the *Borrower name* and the *Current loans* are shown, while under *Current copies on loan* only the *Copy number*, *Shelf mark* and *Title* are still visible. The limitation in functionality pertains to the pop-up menu for copies on loan (normally opened by right-clicking a copy), which is no longer available for the self-service users.



Adjusting the messages

The messages you would like Adloan to present to self-service users are probably different from the standard Adloan messages. Normally, those messages are meant for the issue desk employee, and are in the spirit of: "*Borrower <name> entitlement exceeded for loan category <borrower category>. Continue?*". In the message, the issue desk employee can subsequently choose between *OK* and *Cancel*. A borrower however, may not be allowed to make this choice.

There are a number of other messages which might need rephrasing for the self-service user, and which need to be outfitted with an *OK* button only, to always cancel the transaction.

That is possible, but it has the consequence that you'll have to make

a copy of your Adlib \executables folder (containing *adloan.exe*), in which you can apply some changes:

1. Create a copy of your Adlib \executables folder, place that copy in the Adlib main folder too and call it \adloan selfservice or \executables copy for example (as long as you know what the folder is for).
2. In that copy folder you'll find *adloan#.txt* files for different languages (# stands for a language number). If you are only working in English, then open *adloan0.txt* in a text editor like Windows Notepad.
3. In here you can adjust message texts as well as the type of the messages. The message type is determined by the so-called "severity code". In the commentary at the top of this text file (and also in the manual for this module), these codes are explained. Especially relevant are codes 5 and 6. A message of type 5 displays two buttons: *OK* and *Cancel*, while a message of type 6 only displays an *OK* button, which always cancels the transaction.
It is probably sensible to change some messages of type 5, related to issuing, into type 6, and adjust their text. In the message you could indicate that the borrower is requested to turn to the issue desk for more information about why the transaction was cancelled. Scattered from line 403 up to and including 469, you'll find the relevant messages. For example, you could change "403 6 Error while issuing %2, error %3." into "403 6 Error while issuing %2, error %3. Please turn to the issue desk for more information.", and line 445 could be changed from "445 5 Borrower %1 entitlement exceeded for loan category %2. Continue?" into "445 6 Your entitlement exceeded for loan category %2. Please turn to the issue desk for more information.", etc. Try not to make the message texts too long, by the way. On display, the percent codes will be filled with relevant data by Adloan.
4. Save your changes.

Creating a shortcut

It's best to set up a separate work station for the Adloan self-service. On that work station, you'll always start Adloan Circulation with a special shortcut, so that the login dialog will be skipped, and the self-service user will be logged in automatically.

So, create a shortcut for Adloan Circulation, on the desktop of the self-service work station. In the Installation guide for Museum, Library and Archive, you'll find general information about creating shortcuts. Also available are the following parameters: *-u* (user name), *-p* (password), *-l* (location/branch), *-s* (stand-alone) or *-h* (adloan

server name), and `-timeout` (number of minutes). With `-timeout` you ensure that the *Issues* tab will be emptied automatically after the number of minutes you indicated in the shortcut. The next borrower will then have a good chance to be welcomed by a clean entry form, without the details of the previous borrower. For the stand-alone application of Adloan Circulation you must use the `-s` option, while you must provide `-h` with the server name if you have installed adserver; also see the Loans module guide. The shortcut will have properties similar to the following example (but with full paths) for the `selfserviceuser` with password `pwl`:

Target:

```
"... \execopy\adloan.exe" -s -c 1 -u selfserviceuser -p pwl -l  
main -timeout 1
```

Start in:

```
"... \library loans management"
```

Clicking the shortcut now immediately opens Adloan Circulation in self-service mode, without the need to explicitly log in.

Updating Adlib software

When you ever install an update (new version) of the Adlib software, do remember to copy the updated executables to the copy folder for Adloan self-service as well, since the update procedure does not do that for you. However, do not overwrite your custom *adloan0.txt* file when you copy the update, otherwise the standard messages will return!

3.13.8 Enabling the Shelf mark column

On the *Reservations* tab in Adloan Circulation, in the list of reserved copies, there is a *Shelf mark* column present. Reservations are mostly made on title, in which case a shelf mark won't be available, but when a reservation is made on copy number, it is handy to have its shelf mark available here, for instance if a copy reserved by telephone or website is already present on the shelf and must be retrieved by a co-worker. Although the new column is directly visible (in Adloan 7.1 and higher), it will only be filled with the shelf mark after a reservation on copy number when you've made some changes to the *reserv* database structure, here in Adlib Designer. You can do that as follows:

1. To be safe, create a backup of your database(s) and your Adlib *\data* folder first.

2. In the Application browser of Adlib Designer, open the `\data` folder, and in it the *reserve* database structure definition. Create a new field in the *Fields* list: right-click *Fields* and select *New > Field* in the pop-up menu. For this normal field, provide the tag `ey` and the field name `copy.shelfmark`: that name can be entered for the other languages as well. The maximum length of the field must be set to 64. Leave the other options as they are.
3. In the *reserv* database structure now select the *copy.number (r0)* field and open its *Linked field mapping* properties tab. In the upper list (*Copy fields from linked record*) two mapped fields are present already; a third one must be added here. In the *Source field in copies* column, click the left ... button in the third, empty row: choose the *shelfmark* field (tag `ey`) from the list of fields that opens and click *OK*. Back in the field properties, click the right ... button (in the third row again, now in the *Destination fields in RESERV* column), select the new *copy.shelfmark* field (tag `ey` again) and click *OK*.
4. Save the changes in the *reserve* database structure.

3.14 Properties of screen files

§

3.14.1 Screen file properties

On the *Screen file properties*, *Descriptions* and *Access rights* tabs, which are present when you have selected a new or existing screen file (not to be confused with a screen reference) in the *Application browser*, you may view or set a small selection of screen file properties. All screen file properties however, can be set in the *Screen editor* while editing the screen. See the Helps topics for these properties in the chapter about the *Screen editor* (Screen design) for more information.

The only *unique* property here, on the *Screen file properties* tab, is *Encoding* (see below).

[Click here](#) for information on how to edit properties in general. And [click here](#) to read about how to manage objects in the tree view in the *Application browser*.

Path

This read-only property displays the full path to this screen file.

Encoding

This read-only property displays the type of character set used to encode texts you provide for properties of this object. It does not say anything about the character set used to encode data in. You can change the encoding of database structures (*.inf* files), application structures (*.pbk* files) and screen files (*.fmt* files) simultaneously, with the *Application character set conversion tool* in Adlib Designer.

3.14.2 Screen descriptions

On the *Screen file properties*, *Descriptions* and *Access rights* tabs, which are present when you have selected a new or existing screen file (not to be confused with a screen reference) in the *Application browser*, you may view or set a small selection of screen file properties. All screen file properties however, can be set in the *Screen editor* while editing the screen. See the Helps topics for these properties in the chapter about the *Screen editor* (Screen design) for more information.

The information in the *Descriptions* property here, is the same as on the *Screen* tab in the *Adlib object properties* window for this screen opened in the *Screen editor*. Changes here or there are reflected on the other properties tab.

[Click here](#) for information on how to edit properties in general.

Descriptions

You must provide a brief description of the screen in one or as many languages as you wish: these descriptions will appear on the label of the screen in the running application.

3.14.3 Access rights

On the *Screen file properties*, *Descriptions* and *Access rights* tabs, which are present when you have selected a new or existing screen file (not to be confused with a screen reference) in the *Application browser*, you may view or set a small selection of screen file properties. All screen file properties however, can be set in the *Screen editor* while editing the screen. See the Helps topics for these properties in the chapter about the *Screen editor* (Screen design) for more information.

Here you determine the access rights for this screen file to restrict access to it, dependent on the user (login name in Windows) and its assigned role.

The information in the *Access* property here, is the same as on the *Access* tab in the *Adlib object properties* window for this screen opened in the *Screen editor*. Changes here or there are reflected on the other properties tab.

Click here for information on how to edit screen object properties in general.

Access (Role, Access rights)

Here you may define which *Roles* have which *Access rights* to this screen. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. No access means the screen won't be visible at all; *Read* access allows the user to view data on the screen but not to edit it; *Write* and *Full* access allow the user to edit the data on this screen.

None and *Read* have priority over the unmarked *Read-only* option for a screen, meaning that the *Read-only* option only makes a screen editable in principle, but that you can still exclude certain users from doing that, through access rights. These two access rights also have priority over less limiting access rights set on other levels in the application or databases.

If a role is not linked to this screen, then each user linked to that role has full access by default, unless restricted by other limiting access rights. A user without a role always has full access.

Users are assigned to roles, in the application setup.

Note that setting access rights on screen level is rarely put to use, since there are better ways to prevent certain data from being edited or viewed, for instance by using application identification roles, or setting access rights on database field level (see the reference below for more information).

See also

Security in Adlib

4 Database setup

§

4.1 Databases

A database is a single physical file (with the extension *.cbf*) that contains all data from the records that you enter or import into this database in a running Adlib application. Examples are the thesaurus and the catalogue databases.

In the database setup you define the structure of a database (the database definition) by its properties, in an *.inf* file).

Datasets

A database can be divided into virtual subfiles, known as datasets. These are not separate physical files on your computer, only a subdivision of the one database file. This allows the user to search subfiles separately, or to search a combination of subfiles. You can also screen off parts of a database from certain users by this division. Generally, datasets are used to group together records that logically belong together. For example, you can divide a catalogue database into a books dataset, an articles dataset, a serials dataset, and an audio-visual materials dataset.

In the database setup you define a dataset by its properties.

See also

Accessing the database setup

How to create a new database or dataset

4.2 Fields

Records in a database are made up of fields. In each field, specific data for the record is stored, such as a title or an author, for a record which holds the data for a book for example. The primary identifier of a field is a tag which consists of a maximum of two characters; without a tag there is no field. Fields can also have a name: a field name is language specific and is a more convenient and a sometimes required way of identifying a field. Through the database setup you can define fields by their properties. By specifying fields this way, you

define what's called a data dictionary: a collection of the field definitions in a database.

Although a complete data dictionary is highly recommended, you are not obligated to define the fields in your database this way, with the exception of linked fields. Before the data dictionary got implemented in Adlib, you defined fields where you placed them on a form. This possibility remained present since then, so you could specify fields in the "old" tools ADSETUP (on a form) *and* in DBSETUP (in the data dictionary). In Adlib Designer you can still do this in two ways, either in the *Screen editor* by placing a new tag on a form, or in the database setup using the data dictionary. Fields can also be created from adapls, simply by the FACS declaration of a new tag. The advantages of a data dictionary definition are:

- With the data dictionary you have more options to set for a field (necessary for linking a field, for example), such as the data type and the field length, for example.
- With the data dictionary you can and should name the field. This name will, for instance, appear in field lists in Adlib applications where all database fields are listed (e.g. in the *Expert search system* and in the *Export wizard*). If you don't want the name of a certain field to appear in field lists, you should set the *Do not show in lists* property for it.

With both ways of specifying a field, the field and its contents are always stored in the appropriate database. (The only exception are data dictionary fields of the *Temporary Data type*: fields of this type are typically temporary compound fields that you use on brief displays.)

The content of individual fields in a newly created record is normally empty, but it can also be preset with a fixed text, a time or date, or user name, or can be generated automatically (like sequential object numbers). In the relevant data dictionary field definition in the database setup, you can specify such default values or automatic numbering.

See also

Accessing the database setup

How to create a field

4.3 Multilingual fields

One record can in principle contain data in several languages. This functionality has not been applied in the Adlib model applications though. There are two different implementations of multilingual fields, which you can use to customize your own application. The first one is the preferred method, but only available for Adlib SQL and Adlib Oracle databases; the second method is deprecated for SQL and Oracle, but it's your only option if you want to make a CBF database multilingual.

1. Using XML's extensibility to store translations (SQL and Oracle databases only)

In SQL and Oracle databases, Adlib data is stored in XML format, and because of its hierarchical structure this format can easily be extended. After setting up a field accordingly, it stores translations of a field occurrence in separate XML-tags per language underneath the record node. Tag, occurrence number, language of the data in this field occurrence and the invariancy* of the language are possible attributes in the XML <field>-tag which contains the data in this field occurrence, for example: `<field tag="au" occ="1" lang="en-GB" invariant="true">data in this field occurrence</field>`

Per data dictionary field you simply decide whether it has to be multilingual or not, and you don't have to create any separate field tags. The languages which must be available are set once, application-wide. Now, the language in which the user enters or reads data in a multilingual record is not determined by the application interface language, but by the *Data language* selection. The currently chosen data language is visible to the left in the status bar of the running application. Note that here the interface language may be set differently from the data language.

* Manually, you can assign a so-called invariancy flag to one of the data languages of any field occurrence, by editing the translations of a value in a multilingual field accordingly via the *Edit multilingual texts* dialog in the Adlib application. The advantage of having values in an invariant data language is that you can show this data in the other data languages to make it easier to enter translations. Whatever data you leave or enter into this field occurrence will be stored with the currently set data language. If you do not switch to another data language, then the field occurrence in that language will remain empty and will not be stored. (Whether or not invariant data is displayed or not, depends on the *Options > Hide invariant data* option in Adlib.)

You make a field multilingual this way by marking the *Multi-lingual* option. The user will be able to enter a value for this field (and each occurrence of it) in the running application, in any language that you

set in the *Data languages* list for the .pbk file of this application. Note that if you want to make a linked field multilingual, you have to do the same for the linked-to field and any internally linked fields, otherwise translations cannot be stored. Also, all (externally and internally) linked multilingual fields need to be linked on (forward) reference. In a running Adlib application on a SQL or Oracle database, the user must choose the language in which he or she wishes to enter data in this field, from the *Data language* button submenu in the *Edit* menu, when the cursor is in a multilingual field (this menu only offers the languages set in the *Data languages* list in Designer).

2. Associating multiple tags with one field (CBF; deprecated for other database types)

By associating different unique tags for every language in which translations must be entered to one and the same field name, you create the possibility to store translations of a field value in both a separate tag and under one field name. You can then link an entry field on a screen to either a specific language tag (if data entry must be limited to the relevant language), or to the field name, in which case the contents of the field are displayed in the application interface language selected by the user (via the *Language switch* option). In the latter case this means you can enter and search data in the language of your choice, and create translations in the same entry field (in the same record).

In the definition of a field that you wish to make into a multilingual field, Designer therefore offers, besides the *Field tag* (for the standard language), the possibility to use *Language tags* as well, for every language in which you wish to be able to enter data. (Because English is the standard language, the *English* language tag is the same as the *Field tag*.)

Each language tag must be a unique tag in this database. These tags are used the same way as any other tag, but you should only link an entry field on screen to a specific language tag, a French tag for example, if you mean to enter only a value in the French language in that entry field.

But creating a separate entry field for every language that you mean to use, is quite laborious and not very elegant when building an application. That is why Adlib has added another difference in the way the name and tag of a field are treated for use with multilingual entry. A field **tag** is always unique, for searching as well as for entering data. However, the contents of an entry field linked to a field name with multilingual tags have been linked to the language in which your Adlib application works:

- For searching with a Dutch fieldname or access point for example, (which happens automatically when you start or switch Adlib in/to

Dutch) this means that Adlib only searches the Dutch-language tag for this field.

- When you enter data, it means that when you start or switch Adlib in/to Dutch and you enter data into a multilingual field, this value is automatically assigned to the Dutch-language tag of the field.
- When you build the application, you will have to add several multilingual tags to all fields in the database (which have to be multilingual), and you will have to link each of those fields in each screen of your application to the standard field **name** instead of to one particular tag, to make the application multilingual. This is, assuming that you already have multilingual system texts and field names. Of course, this does not automatically make the records themselves multilingual. You will have to translate the records yourself, and after start-up you will also have to remember to set the application to the particular language (via the *Language switch* option) to enter records in that language automatically.

This functionality is available both in `adlwin.exe` (for most Adlib applications) and in the Internet Server. The XML search result from Internet Server contains a language attribute, so that data and records can be input and output through your website in the language of your visitor. Of course, the languages the Adlib user can choose from are limited to those in which records and system texts are available.

Remarks

- Actually filling in translations in either implementation of multilingual fields is always optional.
- These two implementations are not strictly mutually exclusive, but it is of course highly recommended to only use one multilingual method in your application.

See also

Fields: introduction

4.4 Indexes

Indexes are 2-dimensional tables composed of "pages" which together hold either actual data (copied from a database for this purpose) or reference numbers to the original data in a linked database, both from one or more specific fields, plus the record number of the record in which the indexed value occurs. An indexed value appears as many times in the index as it occurs in the relevant field(s) in records.

The subdivision of a table into pages is only meant to increase searching speed and limit memory use; when using an index you can ignore this subdivision.

The purpose of an index is to be able to search an often used field more quickly than would be possible when searching through the database itself. Now only the field-specific table (actually an index page at a time) needs to be loaded into memory and searched, whilst searching the database can only be done record by record, which takes much longer. In the database setup you must create an index explicitly for each field you want to be able to search through quickly in an Adlib application.

Further, indexes on link reference fields are a requirement for reverse linked fields and linked fields in feedback databases; in Adlib model applications version 3.6 and higher, feedback links are present for all linked fields, and thus all link reference fields have indexes.

There are two special indexes:

- The first is *preref*, which stands for "primary reference". Adlib uses this index to list all the record numbers (= the primary references of each record) from your database together with the location of each record in the database file. The *preref* index is created automatically when a new file (database) is defined, and is therefore always present. This index cannot be deleted or modified.
- Another special index is the *wordlist*. It is created by the database setup in Designer as soon as the first free text index (previously called a word index) is defined. This wordlist index contains all words that "occur" in all free text indexes of all databases in a directory. Adlib uses this index to save memory when indexing words; thanks to this list, when compiling free text indexes, Adlib can suffice with referring to a serial number in the wordlist instead of including whole words. So only the wordlist index contains the actual words, the free text indexes refer only to the wordlist. This process takes place behind the scenes, so you don't need to take it into account when setting up free text indexes.

Under the hood, Adlib uses these indexes as follows:

- When you search a *Record number* access point, just the *priref* index is used to find a record number and the location of the record in the database.
- When you search a term indexed field, e.g. through an access point, Adlib first looks up the search key in the relevant field index to retrieve the record number(s) of the records in which the value occurs. Then those record numbers will be looked up in the *priref* index to find the actual locations of the records.
- When you search an index on a link reference field, e.g. through an access point or when validating input for a linked field, there are three indexes to take into account. In the linked database there is a term index on the relevant field, which holds the actual keys (terms or names) and their record number in that database. Adlib uses the database definitions to find out about this index and then uses it to search for the search key and retrieve the accompanying record number.

Now, for the current database there is an index which lists the record numbers of the mentioned linked terms or names (since a field linked by reference stores only the linked record number of the term it displays), accompanied by the (local) record numbers of the records in the current database in which the relevant field(s) link to said term or name. Adlib uses this index and the retrieved linked record number to find the relevant local record numbers.

Then finally, the locations of those local record numbers are available through the *priref* index.

Note that in an Adlib SQL database the *priref* index is not applied: the local record number is enough to directly locate the actual record in the table.

Using the *stoptabl* file

In old Adlib applications, for searches and free text indexes, a stop table was used to exclude certain often used or short words from being indexed. In present applications the use of this file is still possible but not recommended. If you still want to continue using the stop table, place the *stoptabl* text file (without extension) in the *bin* or *executables* directory of your Adlib folder. In such a file you just enter all words in whatever language - each word on a new line - to make Adlib exclude these words from indexing and searching. The stop table applies to each database in the *\data* directory that contains a free text index.

If you edit the stop table for an existing database, the references

in the free text index may become skewed. To avoid this, a new index must be generated by deleting the *wordlist.idx* file. And all your free text indexes must be reindexed.

In current free text indexes though, all words, however short, are indexed. Only separator characters are stripped out. These are: [;,!@()|{}<>?] and spaces, new lines and tabs. All combinations with concatenators are indexed. For example: l'arbre is indexed as *l'arbre* and as *arbre*.

This makes it easier to find records, but the wordlist index will become larger.

(Re)building an index

After you create an index definition, the index file must be built up, filling it with the appropriate data from the database. In the *Application browser*, when you create a new index by dragging an existing field to an index list, the index file will automatically be built. If you create a new index from scratch in an index list, you will have to build it manually after you've set it up, by reindexing it. With reindexing you build or rebuild an index using the data in the database. This will be necessary if you have modified the index definition, added a new index to an existing database, or if the existing index has become unusable for some reason (e.g. a power cut).

See also

Accessing the database setup

How to create an index

How to (re)build an index

Index properties

4.5 Linked fields

Linked fields are fields containing information from a different database. Adlib retrieves the linked data from a specified database and places it in standard database fields. That means you can process this data on detailed display screens in the same way as data in standard fields. If you put the retrieved data in fields that can be modified by the user, Adlib can write the modified data back to the linked file if you have specified so.

The database in which you want to search for data is referred to as the linked file. The current database, to which you want to link data, is referred to as the primary file.

So for a link you should create a tag in the primary file, which will contain the key data (e.g. a person's name) that a user fills in. This is called the linked field (even though it is located in the primary file). Adlib will use this key data to search the linked file for related or substitutional data.

In the properties of a linked field, you specify:

- the index field in the linked file in which the key data is searched (e.g. the name index in the addresses file). All index values for the search index must be unique in the application. This is called the lookup field (the field you link to).
- for each item of data to be located (retrieved): the tag in the linked database containing the data, and the tag in the primary database to which the data is to be copied (e.g. street, town/city and postal code)
- for each item of data to be written: the tag in the linked file to which the data is to be written, and the tag in the primary file in which the data to be written to the linked file is entered by the user.

You can also specify whether key data that does not occur in the linked file is permitted or not. If the user is only allowed to enter data in the linked field that also occurs in the linked file, this is known as strict validation. With strict validation, Adlib will display a message if unknown key data is entered. If strict validation is not being used, the unknown data is accepted, but the retrieval fields remain empty. Optionally, you can specify that a user is allowed to add key data to a linked file. This is called forcing or adding, in Adlib terminology.

A linked field must be specified as multilingual if the field it links to is multilingual as well.

Linking by reference

You can also allow the user to modify key data in the linked file, and have Adlib automatically adopt the modification for each reference in the primary file. For this, it must be known in the primary file in which record in the linked file the linked data is located. To this end, you define a so called link reference field (a forward reference) for the linked field in the primary file, in which Adlib specifies the record number of the linked record. (If you do not define a link reference field, linked fields only hold key data, and are not automatically updated if you change that key data in the linked file: the linked information occurs both in the primary file and the linked file.) Once the link has been made, the user can edit the key data in the linked file and automatically "update" the linked field to this record, in all

primary records. An important advantage of this method is that the linked information only occurs once – in the linked file. The primary file only contains a reference to the information in the form of the record number; the linked field is filled only then with the key data from a record in the linked file, when it is required (for instance when the user opens a primary record with this linked field).

The main disadvantage of link reference links is that you cannot make compound indexes from this type of linked field.

Forward reference fields should be defined as normal database fields, of the integer data type, and repeatable if its associated linked field is, and a member of the same group (if any). A forward reference field must never be multilingual, even if it is associated with a multilingual linked field.

You should always make an index on the link reference field. Strictly speaking it is only a requirement for feedback databases and reverse links, but it's good practice to make it a rule. Moreover, in Adlib model applications version 3.6 and higher, feedback links are present for all linked fields by default, and not just for links to authority files.

See also

Make a field linked

Linked field properties

Internal links

4.6 Internal links

Internal links define hierarchical relations between terms in one and the same database, usually an authority file like the *Thesaurus* or *Persons and institutions*.

Internal links are typically used for searching and validating terms entered in linked fields in a primary database, making it possible to automatically have non-preferred terms substituted by preferred terms, to always use the right spelling when entering existing keywords, and to search for narrower, broader, and equivalent terms and semantic factors. In the authority files themselves the internal links are used to automatically create the correct reciprocal (mirrored) records when you define a hierarchical relation in one term record, and to validate new terms against earlier saved terms to make sure your authority files won't be corrupted.

The term field in the *Thesaurus* and the name field in *Persons and institutions* database that hold the main keywords in authority records are not linked fields, but normal fields. All the other relational fields in such a database are linked fields, linked to the term or name field. (In the properties for such internally linked fields, enter a dot for

the *Folder* to indicate the current folder, enter "=" for the *Database* to indicate the current database, and mark *Link only first occurrence*.) An internal link setup is not part of the properties of any one field; internal links are listed and created as separate objects in the tree view of the Application browser, and can be found on the same level as fields and indexes in a database.

There are five different types of internal links. The type defines which reciprocal records are to be created if the current type is (implicitly) used when filling in and saving a term record in an authority file. The types are:

- **Hierarchical (type 1)** - defines the relation between the *Term* field, and the *Broader term* and *Narrower term* field, for example: `transportation > motorized vehicles > motorbike`. A broader term encompasses more than the term, while a narrower term has a more confined meaning: the narrow term is a subset of the term, and the term is a subset of the broader term. For each narrower (or broader) term entered in an authority record, Adlib will create a mirrored record, in which e.g. an "original" narrower term becomes the *Term*, and the "original" term the *Broader term*.

In an Adlib application, this relationship allows the user to search generically on a term. (See the Adlib User guide for more information on searching hierarchically.)

- **Related (type 2)** - defines the relation between the *Term* field and the *Related term* field, for example: `transportation <-> traffic`. (Related terms are rather like "see also" references.) For each related term entered in an authority record, Adlib will create another record in which this term is entered in the *Term* field, and places the other related terms, including the original term, in the *Related term* field.
- **Equivalent (type 3)** - defines the relation between the *Term* field and the *Equivalent term* field, for example: `car = automobile`. Equivalent terms are terms that mean the same, but for which you want to use other terms, for whatever reason. Equivalent terms are often used to represent a term in different languages. For each equivalent term entered in an authority record, a reciprocal record is created, in which term and equivalent term switch places. When the user searches on a term in an application, the search is automatically extended to all equivalent terms of this term (if available). So in effect the user searches on the term and all its equivalents.
- **Preferred (type 4)** - defines the relation between the *Term* field and its preferred or non-preferred term in the *Use* or *Used for*

field, for example: *chopper*, use *motorcycle*.

A reciprocal record is created for each *Use* or *Used for* term you enter in an authority record. *Use* contains a preferred term if the *Term* is non-preferred, while *Used for* contains a non-preferred term if the *Term* is preferred.

If the user searches on a non-preferred term in an application, or enters a non-preferred term in a record, the non-preferred term is automatically replaced by the preferred term.

- **Semantic factoring (type 5)** - defines the relation between a (complex) concept term and its semantic factors. The concept is defined under *Term*, and the semantic factors in (the occurrences of) the field *Semantic factors*. For every relationship you build using semantic factoring, the reciprocal records are generated automatically. In the record of one semantic factor for example, the field *Semantic factors of* sums up the concepts (as a repeated field) for which this term is a semantic factor. (Most existing Adlib thesauri do not have such an internal link set up in the application yet. But from Adlib 5.0 this type can be handled. Your application may have to be modified to be able to fill and read the semantic fields (SF and FV) through the application. But if you do not plan on using semantic factors, your application does not have to be modified.)

Internal links on reference

Internal links have the advantage that changes in one term are processed in all hierarchically related terms, after saving the record. Normally this type of field is linked on text (term), i.e. without forward reference, meaning that after saving a term or name record the internally linked terms not only appear in their own term or name record, but also in the authority records from which they are linked. The above mentioned advantage can only exist though, because the text (term) index in such files is unique. If that index would not be unique, then the automatic processing of "mirrored" records would run into problems if identically spelled terms with a different definition would occur. Which record would or wouldn't need to be processed then?

It is usually desirable that the text (term) index in hierarchical databases is unique. But there are exceptions, such as when you want to include terms from other languages in an English thesaurus, and one or more of those terms have the same spelling as English words, but with a different meaning. In such a case you want to be able to work with a non-unique index, whilst keeping the advantages of automatically processed mirror records. Then it's necessary that the internally linked fields in the hierarchical database are made on reference.

An internal link on reference only stores the record number of the linked record in a link reference tag (the *Forward reference* of an internally linked field), and fills the linked field with a value from the linked term or name record only when you open or use an authority record with such a linked field, or export it (with processing links); but each term appears only once in the authority file, which saves you space on your hard disk.

From Adlib 5.0.2 it is possible to create internal links on reference. But within one database **all** internal links must either be on text (term) or on reference; you can't use them both in one database. By default, internal links in Adlib applications are (still) linked on text (term). For building new hierarchical databases it might be an interesting option to apply internal links on reference.

See also

The Adlib User Guide (for more information about all hierarchical relations in the *Thesaurus* and *Persons and institutions* files)

Accessing the database setup

How to create an internal link

4.7 Reverse links

Reverse links (also called backward references) are a general implementation of many-to-many relationships between records. You implement reverse links if you want databases that are linked to each other by means of linked fields with a link reference, to update each other when in records linked terms are filled in, edited or removed, for instance when you add or remove a record, or when you edit, add or delete a keyword in either database.

Let's say you have a books database and a thesaurus, and in thesaurus records you want to list all the books that refer to the current keyword. If you wouldn't use reverse links, said list of books in keyword x would not be updated if you removed a books record or if you added a books record with a link to this keyword; and the list of keywords in a books record would not be updated if you added or removed a keyword record with a link to this book.

So you have to use reverse links if you want two linked databases to automatically and mutually update each other when linked terms are entered in either of both databases.

To implement this functionality you must specify a reverse link in both databases that are linked to each other. You can only provide a reverse link in a database if there exists a link reference tag for the field you link to in the other database, and if there is an index on that link reference field; and the tag you provide for a reverse link in one

database must be exactly the same as the link reference tag of the field you link to in the other database. So with a reverse link you point to the link reference tag of the field you link to, so that the record number of the record from which you link is stored in (an occurrence of) the link reference tag of the linked record in the other database.

So if you would create e.g. four "primary" records that link to one and the same record in the "linked" database, the record numbers of these four records are stored in the link reference tag occurrences in said record in the linked database. (Note that the terms "primary" and "linked" are interchangeable when talking about reverse links, because each database is at the same time primary and linked.)

Let's illustrate this by the following more elaborate example. Say we have two databases, *Objects* and *Photographs*, with the following characteristics:

Objects	Photographs
Stock number: IN	Photograph number: FN
Linked database: Photographs	Linked database: Objects
Linked field tag: fn	Linked field tag: in
Link reference: LF	Link reference: SF
Reverse link: SF	Reverse link: LF

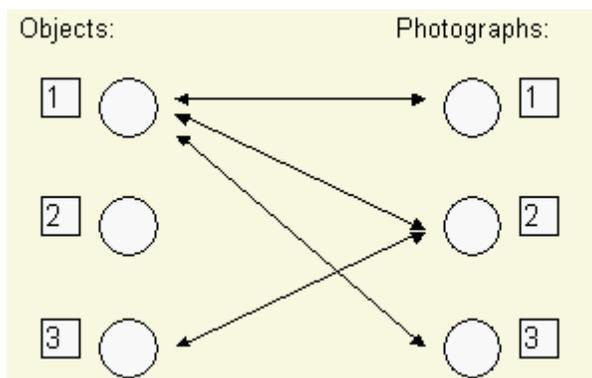
And suppose we have three objects: O1, O2 and O3. Photographs have been taken of two of these objects: F1, F2 and F3. Object O1 is on photographs F1, F2 and F3; object O3 is also on photograph F2. There are no photographs of object O2 in the database. This results in the following tables:

Objects		Photographs	
O1	%O=1	F1	%O=1
fn[1]=F1	LF[1]=1	in[1]=O1	SF[1]=1
fn[2]=F2	LF[2]=2		
fn[3]=F3	LF[3]=3		

Objects		Photographs	
O2	%O=2	F2	%O=2
		in[1]=O1	SF[1]=1
		in[2]=O3	SF[2]=3

Objects		Photographs	
O3	%0=3	F3	%0=3
fn[1]=F2	LF[1]=2	in[1]=O1	SF[1]=1

The above list shows the following information for each of the objects and photographs: the record number (%0); which stock numbers are on which photographs, which photographs the objects are in, and the number of the link reference that ensures the link between object and photograph.



If you click for instance the third photograph number of object O1 in the record for object O1 in the Objects database, you zoom straight to the details of photograph F3. In the same way, you can zoom to object O1 if you are on the linked field "in" of the photograph F3 record. Any modifications made in the objects database that have consequences for the photographs database will be automatically implemented in the photographs database, and vice versa.

So because certain data must appear in both databases, you have to specify that the two databases are linked, and define the link reference tags and reverse tags for the both of them. Then whenever data that appears in both databases is modified, the system can update it in the other database.

Note that some changes, like making a preferred term in an authority file into a non-preferred term doesn't update reverse linked fields to the new preferred term. And if you delete a record with a reverse link, the system doesn't warn you if any references in other databases to this record exist and are going to be removed when you use or open records with those references. For this type of updating you need feedback links, instead of reverse links.

[See also](#)

Linked fields introduction

Make a field linked

Linked field properties

4.8 Feedback links

If you want to be able to make changes in authority records, like making a preferred term in the *Thesaurus* a non-preferred term, or like deleting a name in *Persons and institutions*, and automatically have all records in other databases that are linked to the concerning term or name, automatically updated, you need feedback links.

Feedback links are references in a database definition (often an authority file), to other databases that link to this file. The difference with reverse links is that feedback links can be used in databases that have no linked fields to other databases. Typically, in Adlib applications this would apply to the *Thesaurus* and *Persons and institutions* (although any linked database, even with its own links, may be a candidate): catalogues and other primary databases have linked fields to these authority files to validate the input of terms or names, but there are no fields in those authority files that are linked to the catalogues, because usually you don't need to include data from catalogues in the authority records.

To ensure the integrity of all your (primary) databases, it would sometimes be nice if deletions or changes in term preference which you make in an authority (or other linked) record, would update all records in the (primary) databases that refer to this (authority) record, so that e.g. if you'd make a preferred term a non-preferred term, all catalogue records that would point to this term (through a forward reference) would be updated to point to the new preferred term (another authority record), and that if you'd delete an authority record, you would be warned that there are references in catalogue records to this authority record, or relations with other authority records, and that you are about to remove the record number or key of this authority record from their forward reference fields or internally linked fields. (Note that this type of updating functionality is not provided in reverse links.)

On deletion of an authority record, you will only get one warning that the key appears in a specific catalogue record, even if there are many catalogue records in which the reference appears. However, if you click Yes to remove the reference from the catalogue record, all catalogue records with the reference will actually be updated. If in catalogue records a reference to an authority record is stored in a linked field which is part of a data dictionary field group, and the field group is repeated, then when you remove the authority record, the

reference in the catalogue record field group occurrence will be removed while the other fields in the same occurrence will remain unchanged; only if the other fields in the same occurrence were empty already, will the entire field group occurrence be removed.

However, in most existing Adlib applications prior to model applications version 3.6, feedback links are not being used, even though this may cause problems if you were to make certain modifications to authority records prematurely: for instance, with just forward references to authority files and no feedback links, the record numbers in these references are not deleted when you delete the linked authority record, which in effect corrupts your primary database. Therefore if you do not use feedback links, then in this case you should always first empty all linked fields in the primary databases that contain the authority term that you want to delete from the authority file, and only then delete the authority record itself. For making modifications in existing authority records, similar considerations apply (see the chapter about authority files, in the Adlib User Guide, for more information).

The general advice before making changes to authority records is to first perform an appropriate search-and-replace action in all primary databases, and only then make the corresponding change in the authority record: this prevents your primary databases completely from getting corrupted in any way.

The absence of feedback links in most existing applications prior to model applications version 3.6 has had some good reasons though: for one, all forward references in primary databases that point to an authority file for which you would like to use feedback links, should be indexed. Normally you only make indexes of fields that you want to be able to search quickly, from within a running application. But for feedback links you have to create a lot more indexes, because usually quite a lot of fields are linked to an authority file. Logically, having a lot of indexes decreases performance of Adlib, for instance during saving because more indexes need to be updated, especially for large databases. Another issue with feedback links is that when you actually change a preferred term into a non-preferred term for example, it may take a relatively long time for Adlib to update all primary records that refer to this term. Some terms may be referred to hundreds or even thousands of times, so saving one authority record then means an automatic update of all those records. And this is not indicated on screen with a progress bar, so having to wait it out may be uncomfortable.

However, with the increasing speed of computer processing through the years, these arguments against using feedback links have lost much of their relevance. So if you need some automatic procedure to ensure the integrity of your data (because the order in which primary and authority records should be updated manually, and the

completeness of such an update, is not always carefully observed), it may be a good idea to implement feedback links in your application. In that case, the steps to take for a minimal implementation of feedback links are:

1. Check that all linked fields to authority files, have link reference fields.
2. If a linked field is part of a (repeatable) field group, the field group should have been (or must be) defined in the data dictionary. (In older applications, field groups have sometimes only been defined on a screen, but that is not enough to make the feedback functionality work properly.)
3. Create integer indexes for all those link reference fields (if they do not exist yet), and build them (reindex them), if that didn't happen automatically. (A missing index could result in an error 174 when deleting an authority record.)
4. For each authority file, in the *Feedback databases* list in the database setup, sum up the databases that have fields that link to this authority file.

In Adlib model applications version 3.6 and higher, feedback links are present for all linked fields by default, and not just for links to authority files. This ensures maximum security against database corruption through deletion of linked-to terms or through changes in the preference of terms.

See also

[Linked fields introduction](#)

[Reverse links](#)

[Make a field linked](#)

[Linked field properties](#)

[Feedback database properties](#)

4.9 Domains

You can use so called domains to validate data by logical subsets within a thesaurus that physically only consists of one dataset. Such subsets are known as domains. The advantage of domains is that in one authority file (like a thesaurus) many types of terms can be stored, while terms are validated against specific domains. The way domains work, is best explained with an example. Let's say we have a thesaurus containing topographical names, e.g.

names of countries, cities, streets and districts. The fields linked from the catalogue are *co* (country), *ci* (city), *st* (street) and *di* (district). To be able to validate the name entered by a user when a city is filled in, it is necessary to know whether a topographical name in the thesaurus is the name of a city or of something else (to avoid data corruption). Therefore, all terms in the thesaurus must have been assigned to a domain. When city names like "Rome", "London", and "Amsterdam" have been assigned to e.g. the domain "city", this allows you to search or validate only on city names by specifying this domain.

The example below discusses a situation in which there is a single linked field in the catalogue: city (tag *ci*). This is also an example of a link to a fixed domain name. To make the link to the thesaurus and to the correct domain, i.e. the fixed domain *city* within the thesaurus, certain settings are necessary in both the thesaurus file and the catalogue:

In the thesaurus file (in the database setup)

- **Field settings:** if the domains that can be used in the thesaurus are all known (*country*, *city*, *street* and *district*, in our example) they should preferably be entered in the field definition of the thesaurus *Term type* field as an enumerative list (in the Enumeration values setup). You therefore specify that the field *do* (*Term type*) is an *Enumerative field* with a *Static list* containing the values *country*, *city*, *street* and *district*.
This makes it possible in the thesaurus in a running application to assign one of these four domains to each term; you can then choose that domain from a list in the *Term type* field.
- **Index settings:** first, you must establish that there are domains in the index. This is done in the index in which all thesaurus terms can be quickly retrieved (in this case, the term index). For the *Domain name tag* property in the term index *te*, you enter the tag of the thesaurus data dictionary field in which all the domain names can be found: this is usually *do*: the tag for the *Term type* field in which an enumerated list provides an easy choice between all domains.
An index on terms is necessary when you want to be able to search and validate on terms.
The reason for having to indicate in the term index in what field the domain name for each term is stored, is that in the thesaurus structure there is no association between a term and its domain: both fields merely occur in the same record. Only in an index on terms you can create this association by specifying the domain field. In this index the term field value will then be stored with its domain name as: *domain::key*.

If you add domains to existing thesaurus terms the index on these terms will automatically be updated with the domains, if the index was already defined as an index with domain names. If on the other hand no *Domain name tag* was set for this index, and you now want to be able to search and validate on domains, you'll have to set the *Domain name tag* property, and also reindex this index to include all domains with the terms. (After changing an index definition you'll have to reindex the index to update its contents.)

In the catalogue database (in the database setup)

- **Field settings:** you have to specify that the city field (e.g. tag *sd*) is a linked field, linked to the validation file. This is done through the field properties. For *Name of the domain to link to* (on the Linked field properties tab) the name (*Value*) is entered that was specified as enumerative item in the thesaurus (e.g. *city*). This is a so-called fixed domain. (A distinction is made between capitals and lower case, so the names must be identical.)

Variable domains

The example above was for a fixed domain of a certain field in the catalogue database: for entering a new term in there or searching on appropriate terms for this field, always the domain *city* will be used. When filling in a new city name you therefore cannot choose another domain. City names should just always be stored with the *city* domain.

We speak of variable or flexible domains when you fill in a new term in a catalogue database (not in a validation file like the thesaurus), and you are offered the choice of domain to which the term should be assigned. Usually this will be implemented through an enumerated domain field beneath the term field. The settings in the thesaurus you have to make for such a flexible domain in the catalogue, are no different from those described above for a fixed domain. Only the field settings in the catalogue are different:

In the catalogue database (in the database setup)

- **Field settings:** you have to specify that the term field is a linked field, linked to the validation file. This is done through the field properties. On the Linked field properties tab, mark the *Variable* radio button. For *Tag which holds the domain name to link to* () you fill in the tag of the catalogue field in which the name of the domain will be saved. This catalogue domain field is NOT a linked

field and should preferably be an enumerated list of all available domains for the term field above it. (A distinction is made between capitals and lower case, so the names must be identical.)

In the term index the field value will be stored with the domain name chosen by the user, as: *domain::key*.

The user may also select no domain for the term entered. Just like when you do this in a validation file, the term will just be stored without a domain. Assigning a domain to a certain term, remains optional.

Note that the thesaurus is not the only validation database you can have, and that above examples could also be applied to for instance the *People and institutions* file (in which the *Name type* field holds the domains).

Multiple domains for one term

A single term in a thesaurus or other authority file may be a member of multiple domains. For example, a person's name can be a writer, a customer, and a publisher.

The way to implement this is to make the enumerated domain field a repeated field in the validation file and maybe also in the catalogue (dependent on where you want to offer this functionality). Then the user can add a new occurrence of the enumerated domain field for each domain the term should belong to.

In the term index of this field, each entered *domain::key* combination will be present separately.

Domain-specific icons

It is possible to assign icons to domains. Icons can provide a graphical representation of domains and are especially handy when the user browses through a list of keys that belong to different domains, like when searching on the access point *Term* in the thesaurus. Domain-specific icons are displayed to the left of the concerning terms, in the brief display and in other key lists in an Adlib application. Note that by default there are no icons in Adlib applications.

See the Help topic: Domain-specific icons, for more information.

4.10 Interface functionality for the database setup

§

4.10.1 Accessing the database setup

To access the database setup for your Adlib application in Adlib Designer, you have to open the *Application browser*. With the *Application browser* you can scroll through your application similar to scrolling through folders and files on your computer with Windows Explorer. But in the application/object browser you'll only see folders and Adlib objects, like screens and application definitions and databases. From here, you can add new Adlib objects too, to the folders or lists of your choice. And if possible, you'll find the properties of a selected object in the right pane of the *Application browser*, where you may edit them.

Follow these steps:

1. In the main *Adlib Designer* window that opens when you start Designer, start the *Application browser* by choosing *View > Application browser* or clicking the button for this tool:



2. Select your work folder in the *Application browser* window, by clicking the *Open folder* button:



Preferably, choose your main (copy of an) Adlib folder, not one of the subfolders in it. This allows you to browse all your Adlib objects quickly.

3. In the left window pane of the *Application browser*, click the **+** in front of each folder or object to expand the tree structure and display all objects or folders underneath the current item. Click **-** in front of each folder or object to fold it in. Since all database management takes place on objects in your *\data* subfolder, open just the *data* node. Click any of the Adlib objects underneath the *data* node to display its properties in the window pane on the right; some nodes are just list headers and no objects, like *Fields (#)*, *Indexes (#)*, or *Data sets (#)* and therefore have no properties. These three nodes do display a different list of their objects in the right window pane. These lists offer more information than the Explorer-type lists on the left. Also you can easily sort the list on any of the displayed columns by clicking the column header on which you want to sort, once or

twice (for sorting ascending or descending). For example, sort the datasets list on the *Upper limit* column, and you can easily check if there are no overlapping record number ranges between datasets. You can double-click any field, index or dataset in the list on the right, to open its properties.

Note that underneath the *data* node in the tree view you can see Windows folders names as well as Adlib object descriptions, namely:

-  - folders (to contain the pointer files for a database)
-  - databases
-  - datasets
-  - index list header
-  - indexes
-  - field list header
-  - fields (blue field names denote linked fields)
-  - fields for which access rights have been set up
-  - internal links
-  - feedback databases

See also

Managing databases and datasets

Managing fields and indexes

Editing properties

Saving modifications

4.10.2 Managing databases and datasets

When you have opened the *data* node of your Adlib application in the *Application browser* (see Accessing the database setup), you can edit the properties of each object, but you can also do some object and file management in the tree view. For databases and datasets you can do the following:

Find in application tree

To search for a term in the tree view of the *Application browser*, choose *Edit > Find* (**Ctrl+F**) or click the button for it:



In the *Search for* entry field, type any term or part thereof that you want to search in all of the text displayed in the *Application browser* tree view.

If the term you type is only part of a word or words you look for, then deselect the *Match words* option.

Leave the *Match case* option unmarked if upper and lower case are not important while searching.

Click *Find* to start the search. A found term is highlighted. To search a next appearance of the searched term, each time click *Find next*, press **F3** or click the button for it:



Moving and copying

Parts of an application in the tree view can be moved by cutting or copying a selection and pasting it elsewhere. Right-click an object and choose *Copy* (**Ctrl+C**) or *Cut* (**Ctrl+X**) in the pop-up menu. Then select another list or folder, right-click it and choose *Paste* (**Ctrl+V**) in the pop-up menu. This way, you can copy an entire database and all the objects in it to a different folder. If you copy a database structure to the same folder it originated from, you'll be asked if you want to overwrite the existing file: choose *No* to paste the file under a different name (which you can adjust later).

You can also drag objects from one list or folder to another (to where the mouse pointer displays a +). (Dragging means clicking an object, keeping the left mouse button pressed down, and moving the object some place else, and then releasing the mouse button.) Dragging a database or dataset means copying it. Always drag an object to the header node of a list, to add it to that list. If you drag an object to a similar object **in** a list, that object will be replaced!

Creating new objects

New items can be created. In the tree structure, select the folder or the object in which you want to create a new folder or new Adlib object, and create the object through *File > New*, or right-click the folder or object and choose the new object through the *New* option in the pop-up menu. What objects are available in the *New* menu depends on the node that you selected. For instance, to create a new dataset in a database, you must select or right-click the database name or the *Datasets* list header, not a dataset. Feedback database references must also be created underneath a database node.

Database and dataset names are no longer limited to 8 characters. (But note that if you use longer names, you won't be able to view those names in full in the old DOS tools anymore.)

Creating documentation

In some Designer tools, like in the *Application browser* for a selected database or application, or in the *Import* or *Export job editor*, or the *Record lock manager*, it's possible to generate documentation about the currently selected object or the list of objects, by choosing *File > Create documentation* or clicking the button for it:



A *Documentation* window will be opened with a detailed description of the structure of the selected object. For a database for instance, this information comprises data on the database, its datasets, fields, indexes and links. And the documentation for an application will contain a detailed overview of its properties, menu texts and data sources.

The overview is nicely laid out, but if you rather have the bare XML view, you can switch to it through the *View* menu.

In either view you can print the file via the menu or the *Print documentation* button, or save it as XML file.

You may also select the text, or part of it, with the mouse cursor, and copy (**Ctrl+C**) and paste it (**Ctrl+V**) in any text editor document.

Deleting objects and clearing databases

Select any node in the tree view (a folder, a database, a dataset etc.) that you want to delete and either choose *Edit > Delete*, or right-click the object and choose *Delete* in the pop-up menu, press the **Delete** button on your keyboard, or click the *Delete* button in the toolbar:



Note that deleting any object that has sub-objects, also removes those sub-objects. Folders are deleted along with their contents too, deleting a database removes the database structure (*.inf*), the database contents (*.cbf*), all associated index files, and its pointer files folder. Deletion of files is currently permanent: you cannot restore deleted files from Windows' recycle bin. Deleted sub-objects within the database definition, can be restored by not saving the changes in the concerning *.inf* file when you close Designer.

If you want to clear the contents of an Adlib database (the *.cbf* file or the relevant tables in a SQL or Oracle database) or of all databases in a folder (to remove all data in it), you may do this either by right-clicking the database, respectively the folder and choosing *Clear database* or *Clear all databases* in this folder in the pop-up menu, or select said objects and choose *File > Clear database* to obtain the same result. When you clear a database, all its indexes will be emptied too. When clearing all databases from a folder node at once, the *WORDLIST.IDX* file (which is shared among all databases) will automatically be reinitialised (emptied) too.

Before clearing any database, a dialog will pop up asking for confirmation. Click *OK* to empty the selected database(s), and its/their index files. So all data saved in records will be removed. Your database structure (the *.inf* file) will remain intact.

From Adlib Designer 7.0.0.2 there's also a *Clear dataset* option available, for SQL and Oracle only. The *Clear dataset* option clears the content (all data) of the selected Adlib dataset from the corresponding SQL or Oracle data table and the accompanying index tables and the record access table. The function is available in the context menu for a dataset in an Adlib SQL or Adlib Oracle database only, and compliments the *Clear database* option which clears the entire database. You will be asked for confirmation before the relevant tables are actually emptied.

You cannot undo this action. Click *Cancel* to cancel clearing the database(s) and keep your data.

See also

Managing fields and indexes

Editing properties

Saving modifications

4.10.3 Managing fields and indexes

When you have opened the *data* node of your Adlib application in the *Application browser* (see *Accessing the database setup*), you can edit the properties of each object, but you can also do some object and file management in the tree view. For fields, indexes and internal links you can do the following:

Moving and copying

Parts of an application in the tree view can be moved by cutting or copying a selection and pasting it elsewhere. Right-click an object and choose *Copy (Ctrl+C)* or *Cut (Ctrl+X)* in the pop-up menu. Then select another list or folder, right-click it and choose *Paste (Ctrl+V)* in the pop-up menu. This way, you can copy or move a field, index or internal link to a different folder.

You can also drag objects like fields or indexes from one field or index list to another (to where the mouse pointer displays a +). (Dragging means clicking an object, keeping the left mouse button pressed down, and moving the object some place else, and then releasing the mouse button.) Always drag an object to the header node of a list, to add it to that list. If you drag an object to a similar object **in** a list, that object will be replaced!

Dragging fields, indexes and internal links, means copying them.

Creating an index for a field

There are two ways of creating a new index:

- Right-click a data dictionary field name in the tree view of an opened database* in the *Application browser*, and choose *Create index* from the pop-up menu (only available if the field has no index yet). The type of the field determines what kind of index is created.
If you create an index for a linked field which has a forward reference tag, then the index will be created for the forward reference tag automatically (since the linked field itself does not contain data).
- If you drag a field to an index list header (where the mouse pointer displays a +), automatically an index of this field will be created and built in the concerning list, where you can edit its properties immediately, if they need changing.

From version 6.5.37.11 of Adlib Designer, it is possible to create indexes for an Adlib SQL/Oracle database as well as Adlib CBF indexes.

Creating new objects

New items can be created. In the tree structure, select the folder or the object in which you want to create a new folder or new Adlib object, and create the object through *File > New*, or right-click the folder or object and choose the new object through the *New* option in the pop-up menu. What objects are available in the *New* menu depends on the node that you selected. For instance, to create a new field in a database, you must select or right-click the database name or the *Fields* list header. Similarly, you may create new indexes or internal links.

Important: a new index must be built up by reindexing it - the index file will be (re)filled with data - after you have set all the properties of a blank new index. Reindexing is not necessary if the field for this index does not yet contain any data.

Deleting objects

Select any node in the tree view (a field, an index, an internal link etc.) that you want to delete and either choose *Edit > Delete*, or right-click the object and choose *Delete* in the pop-up menu, press the **DELETE** button on your keyboard, or click the *Delete* button in the toolbar:



Note that deleting any object or node that has sub-objects, also removes those sub-objects. Folders are deleted along with their contents too. Deletion of files is currently permanent: you cannot restore deleted files from Windows' recycle bin. Deleted sub-objects within the database definition (including index files, the exception to the rule), can be restored by not saving the changes in the concerning *.inf* file when you close Designer.

From version 7.1.0.31 of Adlib Designer, it is possible to delete indexes from an Adlib SQL/Oracle database as well as deleting Adlib CBF indexes.

(Re)building indexes

Reindexing of index files can be done by right-clicking an individual index to be reindexed, or a database or folder in which you want to reindex all indexes that appear in it, and choosing the appropriate *Reindex...* option in the pop-up menu. The pop-up menu options *Reindex all indexes* and *Reindex all databases* just index/reindex all indexes (of all types) underneath the selected node. *Reindex all word indexes* just indexes/reindexes all indexes of the *Free text* type,

underneath the selected node. If you reindex all word indexes from a `\data` folder node (and only from there) by means of the *Reindex all word indexes in this folder* option in the pop-up menu, the *wordlist.idx* file will first be deleted from that folder, after which the reindex process will also automatically rebuild the *wordlist* file (although currently the automatic deletion of the wordlist only applies to CBF databases). When using an SQL database, the wordlist should be cleared beforehand manually by deleting the SQL *wordlist* table using SQL Server Management Studio.

You can also select one of said objects and choose *File > Reindex ...* in the menu bar to obtain the same result. Moreover, there's a *Re-index* button available on the *Index properties* tab of an individual index if you just want to index or reindex that particular index.

Note that in general, reindexing forces a save of any modified *.inf* files before reindexing is actually started.

Also, you can run only one reindex job at a time; while reindexing you can continue other work in Adlib Designer though. Do not close Designer until a reindexing job has finished.

You may cut off a running job by clicking the *Abort* button in the progress window; aborting reindexing only stops the process, it doesn't restore the indexes to the state prior to reindexing.

Any reindexing errors will be reported in the main *Designer* window.

From version 6.5.37.11 of Adlib Designer, it is possible to reindex indexes of an Adlib SQL/Oracle database as well as Adlib CBF indexes.

Checking or dumping indexes

The *DBSETUP Check* and *Dump* functionality for analysing the contents and structure of indexes, has not been made available in Adlib Designer.

See also

Managing databases and datasets

Editing properties

Saving modifications

4.10.4 Editing database object properties

If properties of an object are displayed in white or yellow entry fields, you can edit them. (The yellow colour is just for visual presentation, it has no special meaning.)

Field names

Language	Text
English	title
Dutch	tite
French	titre
German	Titel
Custom1	
Custom2	

Just click in the entry field, and delete or type characters.

Logging

Logging file

If properties of an object are displayed in greyed out entry fields, you cannot edit them: they are read-only.

Sometimes you can only choose values from a drop-down list. If such a property reads *Undefined*, it means that for that property the default value will be used: usually the default value is the first value in the drop-down list.

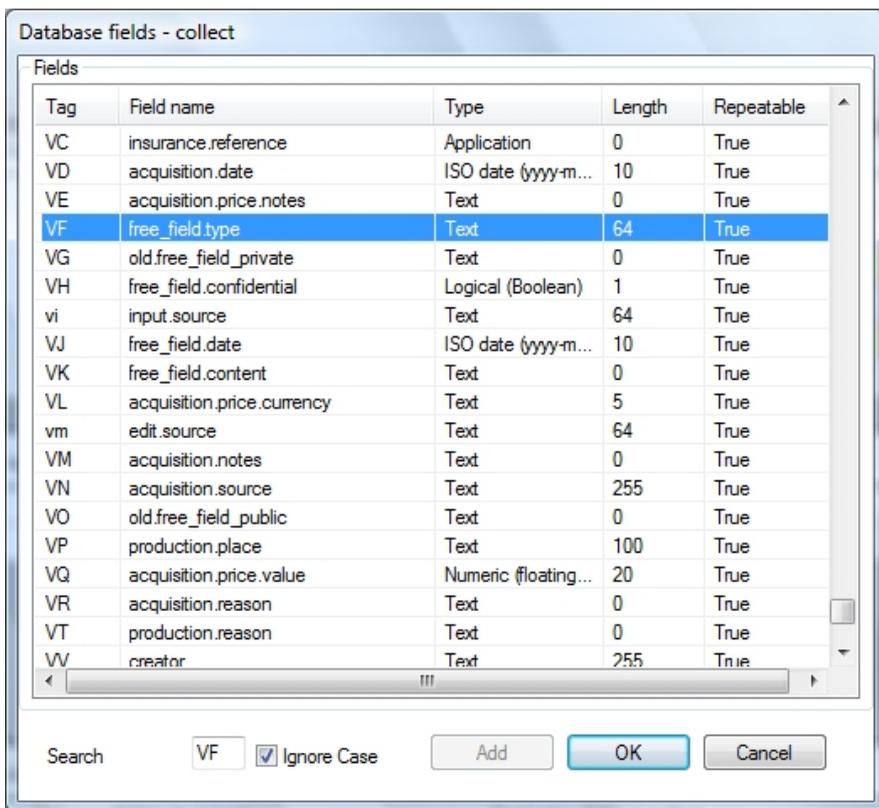
Choosing a value from a list

Behind a number of entry fields, you'll see a grey button with three dots on it (see the figure above). Click this button to either open a list of files/objects that are available for the current property, to search your system for a specific file/object, or to create a new file/object at a specific location on your system. Whenever the button is available, you are advised to use it. For properties that need a tag or field or other object name (like a dataset name), the list offers all the objects that you can enter for this property, and choosing one from the list makes sure the name is spelled just the right way and that any path to the object is entered correctly.

Even though when you type values yourself, sometimes they are validated against the list that you open with the ... button, there is still a chance that by typing you enter incorrect values, either because you forget that certain values are case-sensitive or because you remembered the name of a tag incorrectly. So use the ... button

whenever possible. It helps enormously to be able to just choose from the values that apply for the current property. For example, for the properties of a linked field you can choose the linked *Database* name from a list of database files, and then you can choose the *Lookup field* from a list of fields available in that linked database! Similarly, you can choose your *Forward* and *Backward references* from the appropriate field lists too.

In the list window that opens when you click the ... button to search for a database field, you choose a field either by double-clicking it, or by selecting it and clicking the *OK* button (not visible in the reduced image of that window, below).



You can sort the list on any column ascending or descending by clicking the column header once or twice.

You can limit the list to all tags that begin with a certain character by typing that character in the *Search* entry field. Remove it to display the entire list again. If you mark *Ignore case*, the tags searched with the character(s) that you provided can be upper case as well as

lower case.

If in the *Search* field you type a tag that does not exist yet, and you click *Add*, the new tag is forced in the field property, but no field definition is created in the concerning database, so use this option sparsely, it at all.

Change the name of an object

If you change the name of an object through its properties, the new name will also appear in the current node in the tree view to the left of the properties.

The other way around also works: if you select the name of an object in the tree view, and then click it again or press **F2**, you can edit the name in place. This is reflected immediately in the properties of the object. Sometimes you can change an object name this way, when the object properties show the name as read-only.

But be careful with changing names of properties. Currently, Adlib Designer does not do anything with the new name, even when there are many objects that refer to this object. So you'll have to find all references to the old name of the object in other objects and change them manually. In the future, Adlib Designer may do all this renaming of references automatically.

Moving through properties

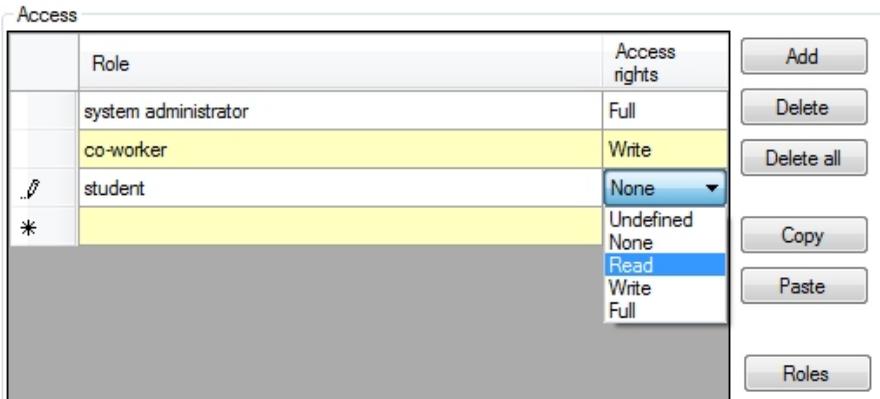
You can easily move downwards through properties by pressing **Tab** on your keyboard, or **Shift-Tab** to move upwards.

Editing mappings

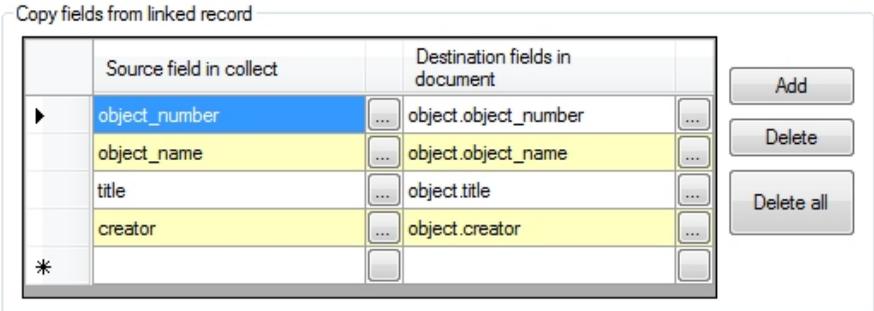
On some properties tabs, like *Linked field mapping*, *Enumeration values* or *Access rights*, you'll find mappings of properties or objects, like combinations of source and destination fields to copy, or assignments of access rights to roles.

The appearance of mappings varies with the version of Designer and with the requirements of the mapping:

- In older versions of Designer (6.3.0 and older) you may see mappings accompanied by a button column to their right side: click the *Add* button to add a new entry in the mapping list; click *Delete* or *Remove* to delete the currently selected mapping entry; if *Up* and *Down* buttons are presents (for e.g. an enumeration mapping) you sort the mapping with these buttons.



- From Designer 6.5.0, you may encounter mappings or lists which always have a new empty line present, indicated by an asterisk in front of it: as soon as you enter a value, a second new line will be created automatically. To enter a value, just click the relevant entry field and start typing, or click the button with the ellipsis on the right side of the field to search for an existing value.



Most mappings in newer versions of Designer (from 6.5.0) no longer have a button column. In this case, all functionality for the relevant mapping is available in a pop-up menu which opens as soon as you right-click a line in the mapping.

When adding a new item or editing an existing entry, you sometimes have to choose options from drop-down lists that appear as soon as you select the property or object in the mapping, or you must choose an object from an objects list in a new window. In general, click the property or object in the mapping to edit it, and you'll be presented with the available choices. Sometimes you may also type a new value. See the Help topic for the concerning property for more information.

Managing roles

On the different *Access rights* tabs in Adlib Designer 6.5.0 or higher, you'll find the *Roles* option in the pop-up menu: right-click a line in the mapping to open the pop-up menu. Choose that option to open the *Current list of roles* window. You are presented with an overview of all roles which are currently in use in your Adlib system. Adding a new role from the *Access rights* tab can only be done in this window: just click the empty entry field and type the name of the new role. Now it is important that you move the cursor to the next empty line after you've entered the new role, otherwise the new role won't be stored in memory. Close the window via the x button in the title bar. The new role can be found in the drop-down list which opens when you want to select a role to assign access rights to.

Note that the new role will only be saved if you actually apply it somewhere in Adlib, to assign access rights to an object. Moreover, the overview in *Current list of roles* will only be refreshed (put together again) when you restart Adlib Designer: redundant empty lines or roles which are no longer in use, will automatically disappear from the list. So you don't need, and can't delete roles manually: a role will be removed from the lists when it is no longer used.

Copying access rights

From Designer 6.5.0, roles and assigned rights set up for one Adlib object, can be copied and pasted to other Adlib objects, even to multiple objects at once. Moreover, copied roles and rights can be edited on a special clipboard, before you paste the access rights to other objects.

Proceed as follows to copy access rights from one Adlib object to others. First, right-click any of the lines with assigned access rights on the *Access rights* properties tab for the relevant Adlib object, and choose *Copy* in the pop-up menu which opens. This copies all access rights to the *Access rights clipboard*, which is really a skin around the Windows clipboard. You can view its content and edit it if you wish. Choose *View > View rights clipboard* in the menu to open the special clipboard. If you are satisfied with these access rights, then close the clipboard via the red-white cross in its title bar. However, you can still make changes first. Just like on the *Access rights* tab, you may edit roles and rights, add new rights through the *Add* in the pop-up menu, delete a selected line with *Delete* or delete all copied rights via *Delete all*, also from within the pop-up menu. After closing the *Access rights clipboard viewer* (if you had opened it), you may paste the copied access rights to other objects in several different ways:

- paste to a single object: open the *Access rights* tab of the other object, right-click the list and choose *Paste* in the pop-up menu. Or

right-click the object in the tree structure of your Adlib system in the left pane of the *Application browser*, and choose *Paste access rights* in the pop-up menu. Or right-click the object in a list in the right window pane of the *Application browser*, for instance a field list, and choose *Paste access rights* in the pop-up menu. Observe that access rights may also be copied or deleted from within such a list, via the pop-up menu.

- paste to multiple objects at once: in an objects list in the right window pane of the *Application browser*, you can select multiple objects. Keep **ctr**1 pressed down while clicking all objects you want to select, or click the first object and keep **shift** pressed down while you click another object to select all in between objects at once. Then right-click one of the selected objects and choose *Paste access rights* in the pop-up menu to apply the copied access rights to all selected objects.

Note that all existing access rights of the target object will be overwritten when pasting!

By the way, it is not necessary per se to copy an object's access rights first. You may also begin by opening an empty *Access rights clipboard viewer*, and then add all desired access rights via *Add* in the pop-up menu.

See also

Accessing the database setup

Managing databases and datasets

Managing fields and indexes

Saving modifications

4.10.5 Saving modifications

Changes* in the properties of an object in the database setup can be saved directly manually in the current database structure (*.inf* file) by right-clicking the database node which the object is part of, and choosing *Save* in the pop-up menu, by choosing *Save (ctr*1+*s)* in the *File* menu or by clicking the *Save* button:



Since all sub-objects in a database are stored in one and the same *.inf* file, you cannot store changes in the sub-object separately; you have to save the database structure as a whole to save the changes to an individual field, for instance.

You can also only save changes in selected files, by choosing *File > Save all...* (**Ctrl+Shift+S**) or by clicking the *Save all* button:



In the *Save objects* window that opens when you click this button, you will be presented with an overview of all the unsaved files in which you made changes, and you can determine of each of these separately whether you want to save them or not, by selecting them or deselecting them. Click *Yes* to save the selected files. *Cancel* returns you to Designer.

When you close Adlib Designer while you haven't saved all changes yet, the *Save objects* window will automatically appear, allowing you to save your work before the application is closed.

Note that in the *Application browser* only physical Adlib objects (files) can be saved separately (currently these are just the *.pbk*, *.inf*, and *.fmt* files). This is because sub-objects like e.g. datasets or fields, are not single files. To save changes in sub-objects immediately, save the larger file that incorporates them, like the database definition.

While managing objects in the tree view of the *Application browser*, there are some actions that initiate an automatic save of an object, such as reindexing. Reindexing forces a save of any modified *.inf* files before reindexing is actually started. The rebuilt indexes are saved when reindexing. Further, databases and folders are saved on creation.

* The nature of properties forms and lists sometimes requires you to leave a property that you just edited, before you can save it. So when you have edited your last property for today, make sure you first click some other property or tab, before you save all your work.

Your *.inf* files are save

To prevent possible loss of *.inf* files because of any errors occurring while you save an *.inf* file, there is an automatic backup procedure in place. When you instruct Designer to save your work, it first makes a backup of the files to be saved by renaming the original files (meaning: the files in which you have not yet saved your current changes). Those backup files have the same name but the extension *.bak*. In the unlikely event that the subsequent saving of the changed *.inf* file corrupts the file, this file will automatically be closed and deleted, and the *.bak* file is given back its original extension *.inf*. Of course you must then enter your changes to the file again, and try to save it more successfully; but at least you will have a proper *.inf* file. But usually saving your files will happen without problems, and when

a save has indeed been successful, the automatically created *.bak* files will be deleted automatically again too, as if they were never there.

4.11 Properties of database objects

§

4.11.1 Database properties

On the *Database properties* tab, which is present when you have selected a new or existing database, you determine the general properties of this database.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Database Name

When you create a new database, you provide the name for it that is displayed here. Adlib Designer will create a new (empty) database file. In addition, a subfolder will be created in *\data*, with the name of the database. Adlib will store any (future) pointer files for the database in this subdirectory.

The data file, index files and definition file for the database will all receive the current file name, with an extension of three characters indicating the file type (respectively *.CBF*, *.OO#*, and *.INF*).

This database name must comply with the rules specified by your computer's operating system for file names, and must be entered without an extension, preferably using only lower case letters (although database names are processed case-insensitively). For use in a DOS environment (DBSETUP), the name has a maximum length of 8 characters. In Windows environments you could, in principle, use long file names of up to 255 characters; but due to limitations in the length of table names in Oracle databases, you should always limit a new database name to 8 characters, even under Windows, to ensure future compatibility in the case you ever move your databases to Oracle.

Path

De database structure that you define on here is saved in an *.inf* file with the database name, in the *\data* folder of the currently opened

work folder in Adlib Designer. The full path is displayed here (read-only).

Encoding

This read-only property displays the type of character set used to encode texts you provide for properties of this object. It does not say anything about the character set used to encode data in.

You can change the encoding of database structures (*.inf* files), application structures (*.pbk* files) and screens (*.fmt* files) simultaneously, with the *Application character set conversion tool* in Adlib Designer.

Authorisation type

Adlib can restrict access to information in various ways. One of them is through authorisation per record per user or role. If you set this property to anything but *None*, then in a running application Adlib will only give a user access to a record in this database if the user has the appropriate authorisation. Adlib will use login data to establish which user is requesting information. In the *Authorisation user field* (see the option below) you can specify per record to which users or roles special access rights apply.

For the current property, you can choose between *Exclude*, *Include* and *Specified rights*:

- If you opt for *Exclude*, all users and roles will have full access to the record with the exception of those named in the *Authorisation user field*.
- If you choose *Include*, only the users and roles in the *Authorisation user field* will have access to the record.
- *Specified rights* makes it possible to specify individual and detailed access rights per user or role per record.

Authorisation field

Here you enter the new name or the tag of the field in which the user names and/or roles that you want to use for authorisation will be filled in, and saved, per record in this database. Depending on the *Authorization type* specified in the option above, these names will be used to determine if and how users have access to the record.

Authorisation rights tag

For the *Authorisation rights field*, fill in a new field name or tag. In this field, per user or role the specific access rights will be stored, that are assigned by the creator of a record.

This option can only be filled in if the *Authorisation type* has been set to *Specified rights*.

Default access rights

The *Default access rights* concern the access rights for all users to a record in which those users or their roles have not been explicitly assigned access rights; this option only applies if the *Authorisation type* has been set to *Specified rights*. If you are more likely to list users in a record, that have extensive access rights like writing or also being allowed to delete, then for this option you'll probably want to set limited default access rights, like read-only or no access at all. The reverse situation will probably occur less often. But it is of course important to choose these default rights carefully, because you won't be listing the majority of users in each new record.

Record owner tag

For the *Record owner field*, also fill in a new field name or tag. Later, in this field the user name or role of the creator of a record will be stored automatically. This name or role is determined through the login of the current user. A *Record owner field* is mandatory for the *Authorisation type: Specified rights*.

For the other two types you may also use a *Record owner field*, but this is optional and does not protect a user names list in a record from being edited by all users with write or full access.

Record owner type

For the *Authorisation type: Specified rights* you can choose here if the name of the *Current user* or the *Current role* (the role of the current user) will be stored by Adlib as the record owner. The advantage of *Current role* (available from Adlib 6.6.0) over *Current user* is that if records are property of a role, you can always assign that role to other users so that they also get the right to adjust the access rights of others to the relevant records: this prevents a potential problem of the *Current user* option, when a particular record owner is no longer in a situation to do his or her work and has not been able to assign a different record owner.

If currently record owners are still user names, and you change the *Record owner type* to *Current role*, then the new setting only applies to new records; the owners of existing records will remain user names. Only those users could use search-and-replace in existing records to replace their name by their role, if desired, so that all records get a role as their owner.

Encryption

Sometimes it is seen as a security risk that an Adlib CBF database can be opened and read in a text editor. (For Adlib SQL Server and Adlib Oracle databases this problem doesn't exist.) That is why from 6.0 it is possible to encrypt your CBF databases, using this option, and by doing so, making them unreadable in text editors.

For your existing databases this option will have been set to *None*, to indicate that their data is not encrypted. If you want to encrypt a database, then first export that database, set the *Encryption* option to *Basic*, and then import the data again. The result is an encrypted database.

Note that exporting always produces a non-encrypted exchange file, regardless of whether the exported database is encrypted or not. Also, you shouldn't see this functionality as a way to fully protect your databases. With this encryption you just make it impossible to read an Adlib database in a text editor.

Minimum record size (in bytes)

In an ideal situation, records in an Adlib database are tightly packed together. However, if something is added to a record, the record will no longer fit in its original place and it will be moved to the end of the database. The empty space thus created is added to the so called freelist, the list of empty spaces in the database. For each successive record that is changed or added, the freelist is consulted first to see if there is an empty space somewhere in the database that is large enough for the record. If not, this record is also placed at the end of the database.

The process of checking and updating the freelist when a record is stored is time consuming. The longer the freelist, the longer this process takes.

For that reason, you can choose a value for the minimum record size. This number represents the minimum record size in bytes. You can enter values between 64 and 4096. The default is 512, but you can change it of course. So by default a space of 512 bytes will be used, also for records smaller than that. This increases the chances of a record still fitting in its old place after being modified, and reduces the necessity of additions to the freelist. Additionally, empty blocks of less than 512 bytes will no longer be included in the freelist.

These actions keep the freelist size to a minimum, thus optimizing the speed at which records are saved. On the other hand, the higher the value set for the freelist, the larger the file becomes. And the larger the file, the longer it takes for the list to be searched and updated when a record is modified.

To benefit directly from altering the minimum record size, you should export and then immediately import the file concerned. Both during the export and import operations, the *Process links* property can be unmarked to increase the speed of the operations. It is important to

mark the *Clear database* property when importing. This ensures that the file is completely rebuilt. If you decide not to import and export, the freelist will gradually become smaller.

Note that there is no maximum record size imposed by Adlib.

Locale

With this option you determine the character set in which your database is stored, and also the sort and search order of your language region, because although different languages can use the same (or mostly the same) alphabet, the sorting thereof can be quite different. And when you want to sort Adlib records ascending or descending, it is important that it happens the way you would expect in your language region.

- *Locale* must be set to *DOS* (at the top of the list), when you use an old database which is stored in the pc character set (DOS/OEM).
Adlib for DOS always used the pc character set, so data had to be converted to ANSI before display in Adlib for Windows, and vice versa. And that is what this setting does. This continues to apply to databases converted from DOS to ANSI, but this setting (the conversion) is NOT necessary for applications that have always been used under Windows.
(This setting does the same as *Convert OEM to ISO* in DBSETUP set to *Yes* did before.)
- *Locale* is set to *ANSI*, when you create a new database, or when you open an existing database that is stored in the WinLatin1 (ANSI) character set (which happens automatically when you've always been working under Windows with Adlib). If you are not interested in the use of characters that do not occur in this set (like Chinese, Hebrew, etc.), then you can just continue working in ANSI.
If you would like to profit from the possibilities Unicode offers, such as being able to enter foreign characters, or to let records be sorted according to your language characteristics, then for you a conversion of your databases might be interesting. This is not just a matter of changing this setting though. It involves an elaborate conversion. Ask the Adlib Helpdesk for more information.
- If this is your start with Adlib then you'll be best prepared for the future if you use Unicode from the beginning. Then choose here the language in which you plan to work, and make this setting for each database. If your databases are still empty, this is all you have to do. If your databases are already filled with Unicode data, but set up for a different language, and you change the

language, then you'll have to reindex all indexes. Note that you only set your language region once. After that, it's still possible to enter characters from other languages in Adlib records, but Adlib will continue sorting according to your own language characteristics.

The value of this option must be the same for all databases in a single Adlib application, otherwise an error will occur.

(A different ANSI character set can be used by including it in a file called *adlib.chr*. If you want to use this option, please contact Axiell ALM Netherlands.)

See also the Help topic: Character set conversion of your data and/or application

Left truncation

CBF - If you want to make it possible to perform left truncated searches in applications that use this CBF database (even if this is a linked database), you'll have to mark this checkbox. When left truncation is enabled, you can search on e.g. **it* and find for instance: *exit, kit, it*, etc.

Note that you don't make it the default way of truncating in this database: you just create the possibility. In the application setup you determine per method whether to use automatic truncation or not, and on what side.

The reason why this possibility of left truncating is optional, is because all term indexes for this database will become almost twice as large. (Every term is also stored spelled backwards.)

After setting *Left truncation* to true, you must reindex all text (term) indexes (NOT the numeric or date indexes); alternatively you can execute an export job and an import job, both processing links. If you also want to enable left truncation searching for word indexes, you must remove the *wordlist.idx* file (in the *\data* folder), and reindex all free text (word) indexes for all databases, including databases in which you have not set this property to true, because the *wordlist.idx* file is shared amongst all databases. (Note that you can set the current option for individual databases: you don't need to set them all. For searching left truncated in linked databases you do need to set this option to true for the concerning linked databases.)

SQL - For Adlib SQL databases, the *Left truncation* option is irrelevant: left, right and middle truncation are supported by default, and can even be used together, for instance to search left and right truncated at the same time. The indexes do not need to be set up in any special way.

NB The user cannot perform left truncated searches on numeric or

date indexes.

Include in full text index

<functionality not implemented yet>

Logging file

Here, you can specify in which file logging data is to be saved. A logging file stores all changes to your data, and serves as an always up-to-date backup. So the Adlib software can take care of logging and recovery, regardless of the database platform on which your Adlib applications run, but SQL Server and Oracle have their own logging & recovery system which offers advantages over that of Adlib. You will mostly want to use Adlib's logging & recovery for *cbf* databases; for Adlib SQL and Adlib Oracle databases you can switch it off (by emptying this option) and use the relevant platform's proprietary procedures.

If you don't enter a name in this option, logging will not take place. You can also use an existing log file, by searching for it on your system. Preferably, specify a logging file on a different drive from the drive that holds your running application, because you don't want a crash of your hard disk with the running application to ruin your backup too.

As soon as you leave this property, a dialog appears, allowing you set this logging file automatically for all other databases. Decide whether you want to set this logging file only for the currently selected database (*No, do not update other databases*), whether you want to set this logging file for all databases in the current folder irrespective of possible logging files already set for those databases (*Yes, update all databases*), or whether you only want to set this new logging file for all databases that have no other logging file assigned yet (*Yes, but only databases that do not use logging yet*). It's best to use one logging file for all databases together.

See Backups, and logging and recovery for more information about using logging files.

Storage type

This option indicates the storage type of the current database. Normally, you use the Adlib proprietary database format (CBF files); in that case this option reads either *Undefined* or *Adlib file share (CBF)*. (*Undefined* defaults to the first value in the drop-down list, in this case *Adlib file share (CBF)*).

If this database has been converted to the Microsoft SQL Server format or Oracle, then the appropriate type is displayed here.

Data Source Name

Only if the current database is part of an SQL Server or Oracle database, its name will be filled in here. So this is the same name that you assigned to the DSN or TNS Service for it.

Note that because Adlib databases are entered as tables in one SQL Server or Oracle database, each database in Designer uses the same DSN/TNS.

Server

Optionally, you can enter the name of the server of the foreign database.

The advantage of doing so (valid from Adlib 6.1.0), is that mentioning the server name here replaces the need to create a DSN on every workstation from which you need to have access to the database on the server.

User name and Password

If you do not use Windows authentication for SQL Server or Oracle, then underneath the DSN or TNS Service you'll also find the user name and password that you set earlier when you created the DSN or TSN: this means that all users together have one user name and password. You can enter that user name and password here.

If you do not use Windows verification, but do not want that the common user name and password are visible in Designer, or if you do not want to use a common user name and password, then for *User id* you may also literally enter the following string: <mustAuthenticate>, and leave *Password* empty, to use the pbk authentication, database authentication or Active Directory authentication that you activated for the application that uses this database. (Click here for more information about these authentication types.)

4.11.2 AdapI procedures

On the *Procedures* tab, which is present when you have selected a new or existing database, you can specify adapIs that have to be executed automatically after a user performs a certain action on this database in an Adlib application. For an existing database some of these procedures will already be set; for a new application you will probably have to write adapIs yourself, before you can set them as a database procedure here. Of course you can add existing procedures or choose different ones whenever necessary.

You can even change the existing adapls to which these properties refer. But do consider that an adapl might be used by several databases, so changing an adapl might have undesired effects elsewhere.

You can link procedure adapls to both databases and screens. Adapls linked to screens take precedence over those specified for databases.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Storage procedure

If Adlib is to carry out an ADAPL procedure each time a record is saved or deleted, then enter the name of that adapl here. The adapl will only be run when the save or delete command is definitive, i.e. after the user has confirmed the action. The adapl will always be run twice, once just before storage (after user confirmation) and once just after storage. To prevent certain fragments of code from being run twice, use the `&1` reserved variable (in an `IF ... THEN` construction) to find out when the adapl is being run: before storage (`&1 = 11`) or after storage (`&1 = 14`).

Adlib will also execute this procedure when importing data through the Import wizard in an Adlib application. However, storage adapls are not executed when data is imported using Adlib Designer or *import.exe*.

If a record is deleted (after confirmation) the storage adapl will also be run. By using the `ADAPL SELECT NO` statement, you can prevent users from deleting certain records. Again, use the `&1` reserved variable as a condition before running deletion-specific code: before deletion (`&1 = 12`) or after deletion (`&1 = 15`).

If a screen is linked to both an after-retrieval adapl and a storage adapl, the after-retrieval is run first, and only when the record is saved (after confirmation) the storage adapl will be run.

Note that records which are stored using the Adlib API (`wwwopac.ashx`) do not trigger this storage procedure.

After retrieval procedure

This is a procedure that is carried out after a record is retrieved. This can be useful if records are being retrieved for a brief display. For example, you could use it there if you want to enforce a certain punctuation with the aid of an adapl, before the records are displayed.

An after retrieval procedure is executed separately for every record visible on the screen: this is relevant if you associate the adapl with a

brief display screen. If you scroll downwards in a brief display screen and new records become visible the after retrieval adapl will be executed for the newly visible records as well. Since the adapl is executed before record data is being displayed, this might have consequences for the performance of the screen build up. So make sure you keep an after retrieval adapl as light as possible.

Copy record procedure

Directly after copying a record, this procedure is executed. Typically it is used to delete unique data in the new record, so that the user is forced to enter new data. An example is the *Copy details* field group in the Library catalogue.

Input record procedure

This is called just before a new record template is displayed on the Adlib screen (e.g. when you click the *New record* button). The procedure is used for example to preset fields with specified values.

Edit record procedure

This procedure is called when you are going to change a record, for example right after you click the *Edit record* button in Adlib, or if you move between tabs when creating a new record. The procedure is also executed before changes are made through an automatic search-and-replace action started by the user.

Field based procedure

This adapl is carried out before or after data is entered or edited in a specific field. You set this in the *Data validation* property of the entry field on a screen, that you want to validate with this field based procedure.

If the screen field (that uses this adapl as an after-field adapl) is associated with a linked field, and the user just entered a different existing linked term in the field and leaves it, then the after-field adapl will be executed before the link has been resolved: this means that when the adapl runs, no link reference or merged-in values are available yet, just the linked value itself.

Also note that a field based procedure won't be executed if the user performs a search-and-replace in the field that uses this adapl.

4.11.3 Advanced

On the *Advanced* tab, which is present when you have selected a new or existing database, you may specify some advanced properties of this database.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Decimal separator

Up until Adlib 6.5.0, numerical values were stored with a dot as the decimal separator. From 6.5.0 this is still the default setting, but you can change that by setting this option to *Comma*. Adlib will notify you that it will have to change this setting for all databases: you may confirm or cancel that procedure.

Note that the presentation format (dot or comma) of numerical values can be specified per data dictionary field, and may be set to depend on the local Windows settings for it. However, the presentation format does not affect the format in print, so if you would very much like numerical values to be printed with a comma as decimal separator, it may be worth considering setting the *Decimal separator* option to *Comma*. Your existing data won't be converted automatically though! After you make this setting, you will have to search-and-replace the dot in all numerical fields in all databases and replace those with a comma.

Further note that the explicit *Default* setting for this option equals the *Dot* setting.

Thesaurus term status field

Enter the field name or tag of the term status field in this database, if the database has such a field.

A term status field can be used to manage authority records by distinguishing approved and non-approved terms in five different categories. See the general topic: Status management of authority records, for more information.

Journal field changes

If you are using Adlib SQL Server or Adlib Oracle, you can mark this option (available from 6.6.0) to let Adlib log all saved changes in records in the current Adlib database. This allows for easy backtracking to see who, when and which changes have been entered, or you can use the change log to find the original data when incorrect data has been filled in. The edit history can be viewed per

field in Adlib, in the *Properties* dialog of a screen field. So the main purpose of this option is to log field data when it changes, so that if you or someone else has changed existing data in a field and saved it, and only later you realize the new data is wrong and you want the old data back, then you can retrieve that old data if you have set this option. See the Adlib User Guide for information about how to do this in the Adlib application.

After setting and saving this option, in the relevant Adlib database table in the Adlib SQL Server or Adlib Oracle database an extra record will be created for every record which has been edited by a user, and this extra record will have the negated number of the original record number: for example, the first change in a record with the number 171 causes the creation of a record with the number -171. In this "negative" record, all* changes in the record will be logged from now on: each data change will be saved in a new field occurrence in this record. Note that the size of your database will increase substantially.

* Only changes in linked fields and any merged-in fields won't be logged.

Store modification history

If you are using Adlib SQL Server or Adlib Oracle, you can mark this option (available from 6.6.0) to allow Adlib to store the creation and modification date and time, and the name of the current user as metadata in an edited record in the current database. This applies to new records as well. To be more precise: this metadata will be stored per edited field occurrence per data language, as attributes of the field node in the XML. For example:

```
<field tag="T9" occ="2" cd="2011-04-11T11:26:51" cu="erik"
md="2011-04-11T11:26:51" mu="erik">Bronski House journey back
to Poland</field>
```

The attributes have the following meaning: *cd* stands for creation date, *cu* means creation user, *md* modification date and *mu* is the modification user. The difference with modification details logged on the *Management details* tab of a record opened in an Adlib application, is that those on the *Management details* tab pertain to the record as a whole, while the current option enables the registration of such details per field.

Note that after setting this option, the size of each new record can be four or five times as large as it would be without this metadata. Metadata quickly adds around 80 bytes per edited field. So if you are about to enter or import a lot of new records, it's good to realize that the size of the database will increase substantially. A large database is not per se a problem related to disk space, the issue is more that

when you retrieve a large record, especially over a network or the internet, it just takes longer to process and lowers general performance. Searching a large database by means of a non-indexed field also takes substantially longer since the database is searched sequentially.

You can't show this metadata in your adlwin.exe application, but you can create indexes on the metadata and define access points for them, so that you'll be able to search for records with fields that have been changed after a certain date and/or by a particular user. You can use this functionality if you want to be able to check all data entered or edited by some user, for example: the access point will get you the relevant records. See the *Metadata type* option for more information.

Access to view candidate terms in link window

The *Show forced terms* checkbox can be hidden from certain user groups via access rights, if you don't want everyone to be able to view and/or link candidate terms. Set those access rights here. The checkbox will be hidden from users with a role which has been assigned the access rights *No*. By default, users do have access to candidate terms. See the general topic: Status management of authority records, for more information.

4.11.4 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing database, you determine the access rights for this database and the objects in it, to restrict access to records, dependent on the user (login name in Windows) and its assigned role. You can also set default access rights for all pointer files together specific to this database; default pointer file access rights for a specific role do not take effect in application data sources for which pointer file methods with any access rights for the same role have been specified.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this database, its datasets and fields. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. If a role is not linked to this database, then each user linked to that role has full access by default. A user without a role

always has full access.
Users are assigned to roles in the application setup.

Default pointer file access rights

Here you may define which *Roles* have which default *Access rights* to pointer files for this database. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. If a role is not linked to this database, then each user linked to that role has full access by default. A user without a role always has full access. Users are assigned to roles in the application setup. If you also specify access rights to a pointer file method in a data source of an application and if that concerns access rights for the same role(s), then those access rights take precedence. Note that the access rights you set here, only apply to pointer files created from now on: existing pointer files are not affected! Also, this setting is independent from the *Access* setting above.

See also

Security in Adlib

4.11.5 Datasets

4.11.5.1 Dataset properties

On the *Dataset properties* tab, which is present when you have selected a new or existing dataset in a database, you determine the general properties of this dataset. Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Dataset name

Enter a name for the dataset. Preferably use lower case letters only: if you don't, know that dataset names are processed case-sensitively.

Lower limit and Upper limit

A dataset is differentiated from other datasets in the database by the range of its record numbers. You can therefore define a dataset within a database by specifying the highest and lowest record number for that dataset. The lowest possible number is 1 and the highest possible number is 2,147,483,647. You should select the

limits in such a way that there is plenty of room for all the records you want to store in this dataset, now and in the future.

Warning: when you divide up a database into datasets, make sure the datasets do not overlap.

4.11.5.2 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing dataset in a database, you determine the access rights for this dataset and the objects in it, to restrict access to records, dependent on the user (login name in Windows) and its assigned role.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this dataset and its fields. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. If a role is not linked to this dataset, then each user linked to that role has full access by default. A user without a role always has full access.

Users are assigned to roles in the application setup.

See also

Security in Adlib

4.11.6 Indexes

4.11.6.1 Index properties

On the *Index properties* tab, which is present when you have selected a new or existing index in a database, you determine the general properties of this index.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*.

Important: you must build up this index by reindexing it - the index file will be (re)filled with data - after you have set all the properties of a blank new index, or when you have modified the properties of an index definition in any way. Reindexing is not necessary if the field does not yet contain any data.

On the current tab you'll find the following settings:

Internal name

Enter a name for the index. This name will only be used within the database setup. If you create access points for an Adlib application, you can use other names. The name may contain an underscore but no full stops, asterisks or punctuation marks.

When you define a new index, it will automatically be given a name; substitute this name with a more appropriate one.

Key type

For an index you must specify its key data type. It sets the way the data in the index file is structured and treated when you search it. It also determines the default sorting order of a search result from this index. For example, if you would like the result of a search on the object number field to be sorted alpha-numerically, instead of alphabetically, make sure the key type of its index is set to *Alpha-numeric*, not to *Text*.

The available index types are:

- **Text (term)** - A text/term index is used if you want to search on the whole field value (or a first part of it), e.g. in the case of keywords. A term index contains keywords in the form of text. The entire content of the field becomes a key value in the index. Any spaces and punctuation marks in the field's contents will be included. If the field is a repeatable field, each repetition will be a separate index key.
- **Integer** - An integer index is used for data consisting of whole numbers, e.g. quantities, record numbers, and such. This type of index may contain integers between -2147483648 and +2147483647.
- **Free text** - You should use a free text index (previously called a word index) if you want to search all the words in a text field separately, e.g. in the case of book titles, when you want to be able to search on each word from the title individually. A free text index contains not the words themselves but reference numbers for the words in the word list (*wordlist.idx*). The word list itself is a text (term) index without fields, containing separate words, without spaces or punctuation marks. All applications in a directory use the same word list. Some words, such as articles or prepositions, have little or no informative value. When you find there is no point in indexing such words or in searching on them, you can exclude them by putting them in the stop table.

- **Date** - A date index contains dates in numeric form, which means that this index does not store the actual date but the number of days since 1/1/1900. This index type accepts dates in the following notations:

European:

dd/mm/yy (e.g. 31/12/94)
 dd/mm/yyyy (e.g. 31/12/1994)

Julian:

yyyy.ddd (e.g. 1994.365)

ISO :

yy-mm-dd (e.g. 94-12-31)
 yyyy-mm-dd (e.g. 1994-12-31)

Only these two variations of an ISO date can be expressed in number of days since 1/1/1900. Other legal ISO dates, like '1984' or '1970-04' can only be indexed using the *ISO date* index type, and only in Adlib SQL or Adlib Oracle databases.

- **Logical** - A logical index is an index on a field which is one position long. If the field contains any character, its contents are considered 'true'. If it does not contain a character, the field contents are considered 'false'. The user does not have to enter a search key for this during searches in Adlib. Adlib will always search for records for which something has been entered in the specified logical field.
- **ISO date** - This type (which can only be applied to indexes in Adlib SQL and Adlib Oracle databases) indexes all legal ISO dates from an ISO date field, like 2009-01-31, 2009-01 or 2009, in a numerical format in which the hyphen between the year and month is substituted by a dot, and the second hyphen is removed, e.g. 2009.0131, 2009.01 or 2009. This behind-the-curtains conversion ensures that ISO dates in this index are always sorted correctly. Also make sure that the *Locale* option in the *Database properties* has been set, otherwise the index cannot function properly.
- **Numeric (floating point)** - <no Help text available yet>
- **Alpha-numeric** - If a field contains a combination of text and numbers and you want the index to sort the text part of a value alphabetically and the numbers numerically (because you want a search result sorted that way), then this index type may be the solution. For example, if two values "a10" and "a9" would be sorted alphabetically only, then a10 would come before a9

because 1 comes before 9 in the alphabet, but if sorting is alpha-numeric then a9 comes before a10 because 9 is numerically smaller than 10. To achieve this, each alpha-numeric number in the index is reformatted so that every number in the string becomes a ten-digit number starting with zeroes. For example, when in a record the string A238HJK08aa2 has been registered, the index value (of which the user can remain unaware) becomes A0000000238HJK0000000008aa0000000002. Note that if the field you wish to create an alpha-numerical index for, contains values with many letter-number alterations, the index values can become pretty lengthy and therefore you should set the *Key size* option of this index sufficiently high: for CBF databases this is no problem but the key size for an index in an Adlib SQL database is limited to 100. So if you employ an Adlib SQL database and you estimate that any reformatted alpha-numeric strings might exceed 100 bytes, then it's best not to create an alpha-numerical index for it and just accept alphabetical sorting.

- **Double meta phone** - In a Double MetaPhone index, the entire field content is indexed as one key value, encoded according to its English pronunciation, so that of an incorrectly spelled search key the correct variant can still be found as long as the pronunciation is the same. It is not possible to search truncated in this index. See the description of the *Phonetic index type* property for more information.
- **Stripped** - This index type was originally introduced to be able to index ISBNs without any concatenators or separator characters in it. So, only the "stripped" version of field content is indexed in an index of this type. For example, if a field contains the ISBN: 90 272 2167 7, it will be indexed as: 9027221677. This makes sure that ISBNs or similar data in this index are always sorted correctly.

Note that not defining an index type is an application error; nonetheless, an undefined index type will be interpreted as a **Text** type index (from Adlib 6.2.0).

Date completion

This option appears if the *Key type* for this index has been set to *ISO date*. Use this option to complete partial dates when they are indexed - a partial date would be just a year, or the year and a month. This is particularly relevant to date range searching in Adlib SQL and Adlib Oracle databases (see the Date range search method type), for which indexed dates need to be complete dates, but you can use it for other purposes as well.

In the case of a date range, an incomplete start date (only a year or year and month) must be equalled to the first day of the year or

month, while an incomplete end date must probably be equaled to the last day of the month or the year. Set the current option for the index of the start date field to *First day of period*, or set it to *Last day of period* for the index of the end date field. For example:

Start date in record	End date in record	Start date in index	End date in index
1960	1960	1960-01-01	1960-12-31
1950	1952	1950-01-01	1952-12-31
2000-03	2000-03	2000-03-01	2000-03-31

Do this for both the start date index and the end date index of each date range.

Also, the date indexes for a date range should only index the first occurrence of the relevant date fields. So mark the *First occurrence only* option in the properties of the relevant indexes.

Reindex the adjusted indexes to apply the changes.

Tags to index

There will usually be a one-to-one relationship between an index and a field, in which case one specified tag is all you need here. Enter the tag of the Adlib field from which this index derives its values in the white entry field (not in the list below it). The tag may be a maximum of two characters long.

You may also index a field linked on forward reference. In that case, the key data field in the primary database cannot be indexed, because the data for the key field is not saved in this database, only retrieved temporarily. Instead, you can define an index (of the integer type) for the forward reference field. (The only place where you refer to this indexed forward reference tag instead of the linked field itself, is when you create a method/access point for the concerning linked field. In screens you always associate an entry field with the linked field name or tag, never the forward reference tag.)

Indexing merged-in fields might be problematic, since the value of a merged-in field is only present in a record when that record is being edited: it is retrieved from a linked record and displayed with a linked field in the current, main record, but not stored in this record. Indexes are updated each time a record is saved, which means that an index on a merged-in field may be built up correctly as long as it is being updated after a main record has been edited and saved, but not whenever the linked record is edited and saved (because then the main record with the linked field and merged-in field won't be opened and saved by Adlib): in this case, the index on the merged-in field will keep the old value until the main record is opened and saved again in some way. If the index on the merged-in field is ever reindexed from

within Designer, it will even result in a completely empty index, since the main records have never stored any merged-in values. So it's best not to create an index on a merged-in field. However, there is a way to create an access point for a merged-in field: see the *Search* key property of methods for more information.

Further, there is a possibility to search on a combination of fields, e.g. by using a combination of title and abstract. To implement this, you must specify a unique tag in the white entry field, that does NOT occur as a field tag in the database yet or that you created specifically already to document the new index tag*. You then *Add* the tags of the database fields to be indexed together, to the grey list below the white entry field.

Just as with the other indexes, you have to specify the main index tag as the *Search field* for the appropriate method (access point) that you specify in the application setup.

For example, suppose you want to allow users of your public access catalogue to search a number of fields with one search command, fields such as title, series, annotation and abstract. To make such a "free text" access point, you would define an index tag *vt* for instance, of the *free text* type with the *Tags to index*: *ti*, *re*, *an* and *sa*. (The tags in this example may be named differently in your application.)

Note that you should only combine fields of the same type (e.g. *Free text* fields) in one index. And a combination of forward reference tags is only allowed if the linked-to fields are located in the same database.

* If you wish to index a combination of fields, it's a good idea to create a separate tag in the data dictionary for the new index tag that you provide for the current property, with the same tag name as that index tag. This reduces the risk of using the index tag for some other purpose too.

Reindex (button)

To make it easier to index or reindex an index after you created a new index or after changing an existing index definition, a *Reindex* button is present next to the *Tags to index* list.

Domain name tag

Here you enter the tag in which the domain names belonging to each term or name to be indexed can be found in this database (often this is *do*). This applies only to indexes that will contain terms belonging to different domains, so indexes on certain fields in validation databases only (like *term* in the thesaurus). The appropriate domain is then added to each field value in the index and makes validating and

searching on domains possible.
(If searching or validating on domains will not be necessary, you needn't fill in any *Domain name tag*.)

First occurrence only

Here, you indicate whether only the first occurrence must be indexed (mark this checkbox), or all occurrences (leave this checkbox unmarked).

Unique keys only

When entering a record, on leaving a field which is indexed (or when saving the record) this option checks whether the entered value already exists in another record (by checking this index). If it does, you will see the message: *This term already exists, only unique terms permitted*. Mark this checkbox to set unique indexing. Leave this checkbox unmarked, if keys in this index needn't be unique.

If the indexed field is multilingual, an entered value must be unique in the current data language; a term can appear more than once in this type of index only if the term has been entered in different data languages. For example, in a multilingual field, the term "Adlib" can be registered in every data language available in the application, and still be indexed with the *Unique keys only* option set.

If a unique index is not necessary but you do want to check whether the index value already occurs in the record you are working on (in the case of a repeated field), you must specify the screen field property *Repeatable* for this field as *Repeated Unique*, in the Screen editor.

Page size (in bytes)

The number of keys per page is determined by the *Page size* and the *Key size*. The index page size (in bytes) is fixed per index. In most cases, the default value (512) will be best. For very short keys, a small page size may reduce the number of disk accesses for a search, resulting in faster retrieval. For very long keys, a larger page size may do better on your system. However, a fixed rule cannot be given.

Key size(in bytes)

This value only applies to a text/term index. The length of a text (term) index key is a minimum of 1 and a maximum of 255 characters (bytes).

You can set this value the same as the *Maximum length* specified for the field of this index, but for terms within a domain you must specify a larger *Key size* because in the index not only the key is stored, but

in front of it also the domain to which the key belongs (separated by two colons).

On the other hand you can also specify the *Key Size* to be shorter than the *Maximum length* of the field, if you would want to index only a first part of the text in this field.

Designer will allocate the correct length for the integer, date, free text and logical index types automatically.

Note that indexes in Adlib SQL databases cannot have a key length greater than 100 characters. If you specify a greater key size anyway, and you try to save a 100+ character term in a record, an error 29 will occur. The record will be saved anyway, but the relevant term index won't be updated. An example of a long term index is *TX* (a term index on the *Title* field in the DOCUMENT database). This key length limitation does not apply to CBF databases.

Phonetic index type

From Adlib 6.5.0, indexes on non-linked fields (in CBF as well as in Adlib SQL and Adlib Oracle databases) can be set to the phonetic index type *Double meta phone*, to allow for phonetic searching in such an index. Phonetic searching means that words can be found which sound the same as the word you are searching for, even when your search key is spelled the wrong way.

The Double Metaphone search algorithm is a phonetic algorithm, developed by Lawrence Philips. With a phonetic algorithm, words are indexed encoded according to their pronunciation, so that of an incorrectly spelled search key the correct variant can still be found as long as the pronunciation is the same. (Equally sounding words share the same phonetic code in the index.) Phonetic algorithms are therefore used in many spell checkers. Double Metaphone is suited for most words and names in the English language: encoding of terms to be indexed follows English pronunciation rules. The use of this search algorithm in other languages may yield unexpected search results.

To prepare the current index for phonetic searching, first set the index *Key type* property on the current properties tab to *Double meta phone*. Since in principle different types are possible, you have to specify this type in the *Phonetic index type* drop-down list - the default setting of this option is *None*. Currently however, this list only contains the *Double meta phone* type, but in the future more phonetic types could be added. Now, set it to *Double meta phone* too.

Subsequently you must reindex this index and save the changes in the database. Only term indexes on non-linked fields are suitable (for example the index on the *Term* field in the *Thesaurus*).

For the relevant access point in the *Search wizard* of a running

application, and only* when your application runs on an Adlib SQL or Oracle database, the user must now choose via a checkbox whether he or she wishes to use phonetic searching or not: after all, the search result may increase, and it may have a negative impact on the speed of searching. If you are sure about the spelling, you can leave phonetic searching off.

* A phonetically encoded index for a CBF database only contains the phonetic codes; the user will automatically search phonetically in such an index, and cannot switch it off. A phonetically encoded index for an SQL or Oracle database on the other hand, contains a column for the normally indexed terms and a column for the phonetically encoded indexed terms. So here, the user does have the possibility to switch phonetic searching on or off.

Metadata type

From 6.6.0, the creation and modification date and time, and the name of the current user will be stored as metadata in an edited record, if you use an Adlib SQL Server or Adlib Oracle database. To be precise: the metadata will be stored per edited field occurrence per data language, as attributes of the field node in the XML. To activate this functionality, set the *Store modification history* option per database.

For example:

```
<field tag="T9" occ="2" cd="2011-04-11T11:26:51" cu="erik"
md="2011-04-11T11:26:51" mu="erik">Bronski House journey back
to Poland</field>
```

The attributes have the following meaning: *cd* stands for creation date, *cu* means creation user, *md* modification date and *mu* is the modification user. The difference with modification details logged on the *Management details* tab of a record opened in an Adlib application, is that those on the *Management details* tab pertain to the record as a whole, while the current option enables the registration of such details per field.

You can't show this metadata in your *adlwin.exe* application, but you can create indexes on the metadata and define access points for them, so that you'll be able to search for records with fields that have been changed after a certain date and/or by a particular user. You can use this functionality if you want to be able to check all data entered or edited by some user, for example: the access point will get you the relevant records.

Leave the *Metadata type* option to *None* for all normal indexes, but for an index on field metadata you must choose between:

- *Creation date*: date on which a value was entered into this field

for the first time.

- *Modification date*: date on which a value in this field was changed.
- *Creation user*: the name of the user who entered a value into this field for the first time.
- *Modification user*: the name of the user who changed a value in this field.

The *Key type* of the index must still be set accordingly (before you can choose the metadata type), so that user names will be stored in a *Text* index and dates in an *ISO Date* index. For the tags to index you must provide a new tag (actually a dummy tag which you should define in the data dictionary too*) followed by one or more tags of the fields of which you want to index the metadata. So if you want to index any of the metadata of the *te* tag (*Term* field) for example, then in the *Tags to index* you first list your new dummy tag (*x1* for example) and secondly *te*. Choose a *Metadata type* to select which metadata of *te* you wish to index and set the corresponding *Key type*.

* It is possible as well to index just the tag of a field with metadata, without using a dummy tag as the first tag to index, but since a field tag can only be used once as the first tag in an index, it's best to use the construction explained above.

For a second example, define a new index of the *ISO Date* type. The metadata available for indexing would then be the modification date and the creation date; choose either one in the current option. Provide two tags: first a new tag of the data type (and presentation format) *ISO date* and secondly the tag of which you want to index the metadata. This means that only when the contents of the second tag in a record has been changed, metadata will be created for it, of which the date type selected here will be indexed in the current index.

As the *Search key* for an access point on this index, you would enter the first (new) tag.

Metadata of changes in linked fields is saved in the link reference tag, not in the linked field itself. So, for the second or further tag in an index on this metadata you'll have to provide the link reference tag. As mentioned, it is possible to create an index on multiple fields, even on dozens of normal and link reference tags of different types of fields: after all, you will only be indexing a single metadatum of those fields.

A change to field contents which is performed by an *adapl*, will also be registered in the metadata.

An example of an application is a web application which retrieves data from a regularly updated copy of your live database, which

displays only a selection of fields to the user. You will always want to update the database copy with just those records in which at least one of the values in the selection of fields has changed after the date of the previous copy. In that case you do not want to search on the edit date of the record, because you are only interested in changes in fields that are relevant to the visitor of your website. After you've created the new index, you can enter such a query in the expert search language of Adlib, save it in a pointer file and renew (*Profile*) it regularly, for instance `z1 > 2011-06-26` (if `z1` is the indexed, new dummy ISO date field), after which you can export these records. Or use the webopac to retrieve the relevant records. The `wwwopac.ashx` request below for example, can actually be entered in your browser's URL box and be executed. The index on the `fields_modification_date` field keeps track of the edit date of no less than 24 fields in the collect database of our API demo application.

```
http://test2.adlibsoft.com/api/wwwopac.ashx?  
database=collect.inf&search=fields_modification_date>'2011-06-  
26'
```

4.11.6.2 Advanced index properties

On the *Advanced index properties* tab, which is present when you have selected a new or existing index in a database, you'll find some read-only properties of this index, mainly for advanced users.

On the current tab you'll find the following data:

Internal name

This name is copied from the option with the same name on the *Index properties* tab.

Index file

This property displays the name under which this physical index file is saved in your *\data* folder.

Physical type

The data encoding type of this index on programming level.

Physical size

The size of the above mentioned physical type of this index in bytes.

Reindexing required

You must build up an index by reindexing it after you have set all the properties of a blank new index, or when you have modified the properties of an index definition in any way. To help you remember if you already reindexed this index or not, after you modified its settings, this option will be marked if you haven't reindexed yet.

4.11.7 Fields

4.11.7.1 Field properties

On the *Field properties* tab, which is present when you have selected a new or existing data dictionary field in a database, you determine the general properties of this field.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Field Tag

Here, you enter the tag of the field which properties you want to describe. The tag consists of a maximum of two characters, of which the first may be a letter, a digit or a % sign, and the second a letter or a digit. Adlib distinguishes between upper and lower case letters. '%0' is always reserved (for the record number). Tags must be unique in the current database: Adlib Designer will check the tag you entered and warns you if it already appears in the data dictionary of the current database, forcing you to enter a different tag.

Type

A field can be any of three basic types: *Normal field*, *Linked field* or *Enumerative field*. You can set normal and linked fields to be automatically numbered as well, by marking the *Automatic numbering field* checkbox.

- Linked fields retrieve their data from another database.
- The information in an enumerative field can either come from a static list that you define beforehand, or a local field.
- A normal field is a field that is not a linked field nor an enumerative field. A link reference tag is also a normal field.
- Automatically numbered fields retrieve their data from an automatic routine. From 6.5.0, linked fields can be automatically

numbered as well as normal fields. An example of a linked field which you might want to have automatically numbered, could for instance be *copy.number* (tag *ex*) in the *DOCUMENT* database. Copy numbers which you create from within the catalogue, will then be automatically numbered according to the settings that you provide on the *Automatic numbering* tab. If you implement such changes, the linked-to field itself (*copynumber* in *COPIES*, in this example) cannot be an automatically numbered field as well. Also any other copy number fields in other databases which link to *copy.number* in *COPIES* cannot be automatically numbered if you already did so in *DOCUMENT*.

Note that a field cannot be linked and be enumerative at the same time.

As soon as you choose the type of this field, extra tabs may be added to the properties tabs to further set up this specific type of field.

- The extra tabs for a linked field are: *Linked field properties*, *Relation fields*, *Link screens* and *Linked field mapping*.
- The extra tab for an enumerative field is: *Enumeration values*.
- The extra tab for an automatic numbering field is: *Automatic numbering*.

Field names

For each language in which your application must be presentable, you must provide a field name here. You can use this name instead of the tag in searches using the expert search language in a running application, and you need field names for export jobs. For example, you could use the field name `author` instead of the more cryptic tag `au`. The field name may be a maximum of 32 characters long and may not contain any spaces. Further restrictions to field names apply because (English) Adlib field names are also used as XML tag names in XML documents produced by the Adlib `wwwopac` (API) or the export function of Adlib. For XML tag names, and thus for Adlib field names, the following practical guidelines can be given:

- The first character of the name must be a letter (a-z or A-Z). Disallowed initial characters for names include digits, diacritics, the full stop and characters like `%&!*+/{ }()[<>-`.
- Names should not begin with "XML" or "xml".
- Beyond the first character, characters permitted in names are letters, digits, hyphens, underscores and the full stop. Not allowed are characters which either are or reasonably could be used as delimiters, like the colon, the semi-colon, a comma,

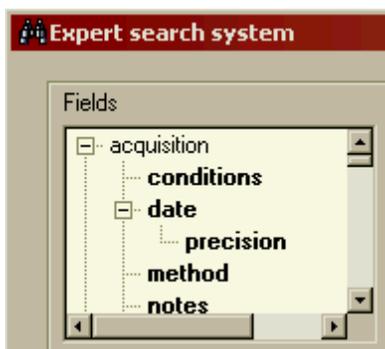
parentheses or brackets for example.

Note that this list is not complete, but English field names usually have no need for exotic characters.

One existing conflicting field in many Adlib applications is the temporary `[display_only]` field in *Document*. The brackets in its name may prove problematic when processed as an XML tag name by an XML parser. However, because it is a temporary field, filled by a before-screen adapt, it is very unlikely to cause problems because it won't be included in records retrieved via `wwwopac`. If the field still causes a problem anyway, you can change the English field name into something without brackets: `display_only` for example. The only other place where the original field name might be in use is in Word templates; so check your Word templates for the appearance of `[display_only]`, and change it to the new field name.

In existing Adlib databases you might find field names with an underscore to separate individual words in the name, such as: *creator_qualification*, since spaces in field names are not allowed. Or you might find it as *creator.qualification*.

The underscore has no special meaning and when creating a field you can use any other character if you find it more clear. The dot-character however, does have a special function: it not only works as a word separator but it also creates a hierarchical order between the name parts, to display field lists in Adlib applications as a tree structure, making it easier to find the field name you're looking for, because field names that belong together are placed together in the list. The *acquisition.date.precision* field can be found in, for instance, the *Expert search system* by opening the *acquisition* root and then *date*:



These hierarchical field lists will also appear in the *Sort* window, the *Replace* window and the *Expert* wizard.

See also the Help topic: Naming fields for hierarchical display.

Maximum length

The maximum length that the field may have. You can enter a value between 0 and 255. If you enter 0, Adlib will enable word-wrapping for the field. In other words, when text is entered in the field, Adlib will automatically move a word to the next line if it no longer fits on the current line. This makes it possible for a field to contain far more than 255 characters.

In the Screen editor, you can only visually specify the length of a screen entry field (a textbox). Since different characters (and different fonts) have different widths, you cannot set the length of a screen field to match the data dictionary field length. So you'll have to experiment to find out if data in the maximum field length always fits into the textbox. It isn't too important though: in entry fields that are too short, you can scroll the contents left and right.

The table below shows the effects of different sizes of the field length and of the textbox.

Field Length	Textbox	Effect
0	any	Automatic word wrap.
smaller	larger	You can't type up until the right edge of the textbox.
larger	smaller	Field content scrolls horizontally across the screen field.

Repeatable

Mark this checkbox if you want this field to be able to have more than one occurrence (occurrences are numbered instances of the same field). For example, because a book may have more than one author, the author field must be repeatable so you can enter each author in its own occurrence of this field.

To be able to actually add new occurrences of an entry field on a screen in a running application, the *Repeatable* property for that entry field must be set to *Repeated* or *RepeatedUnique*.

If the screen field is set to *NotRepeated* while the data dictionary field is *Repeatable*, the screen only displays the first occurrence of the data dictionary field, but the data dictionary field may hold more occurrences (e.g. from an import). The other way around, when a data dictionary field is set to non-repeatable, while the screen field is

repeatable, the data dictionary definition takes precedence: the screen entry field cannot have more than one occurrence.

Data type

Here you can choose from a list of possible data types to specify the type of data that is to be saved in this field. The following types may be selected:

Data type	Meaning										
<i>American date (mm/dd/yyyy)</i>	Can only contain dates like 01/31/2002.										
<i>Application</i>	<p>If you give a field this data type, you can enter a file name in this entry field in the application, including its path. This path is then shown underlined in Adlib to indicate that the field is a linked field. When the user clicks the underlined link, the application that is associated with the file extension in Windows will be started, and the file will then be opened in that application.</p> <p>As soon as you choose this data type for a normal field, the extra tab <i>Application field properties</i> is added to the properties tabs to further set up this specific type of field.</p>										
<i>Date (general)</i>	<p>This date field will accept a date in five possible formats. The following notations are allowed:</p> <table border="1" data-bbox="306 948 948 1145"> <tbody> <tr> <td data-bbox="306 948 642 986"><i>(EUR) dd/mm/yy</i></td> <td data-bbox="647 948 948 986">(e.g. 31/12/94)</td> </tr> <tr> <td data-bbox="306 992 642 1031"><i>(EUR) dd/mm/yyyy</i></td> <td data-bbox="647 992 948 1031">(e.g. 31/12/1994)</td> </tr> <tr> <td data-bbox="306 1037 642 1075"><i>(ISO) yy-mm-dd</i></td> <td data-bbox="647 1037 948 1075">(e.g. 94-12-31)</td> </tr> <tr> <td data-bbox="306 1082 642 1120"><i>(ISO) yyyy-mm-dd</i></td> <td data-bbox="647 1082 948 1120">(e.g. 1994-12-31)</td> </tr> <tr> <td data-bbox="306 1126 642 1165"><i>(Julian) yyyy-ddd</i></td> <td data-bbox="647 1126 948 1165">(e.g. 1994-365)</td> </tr> </tbody> </table> <p>The European date format dd-mm-yyyy is accepted as well, but is now deprecated.</p>	<i>(EUR) dd/mm/yy</i>	(e.g. 31/12/94)	<i>(EUR) dd/mm/yyyy</i>	(e.g. 31/12/1994)	<i>(ISO) yy-mm-dd</i>	(e.g. 94-12-31)	<i>(ISO) yyyy-mm-dd</i>	(e.g. 1994-12-31)	<i>(Julian) yyyy-ddd</i>	(e.g. 1994-365)
<i>(EUR) dd/mm/yy</i>	(e.g. 31/12/94)										
<i>(EUR) dd/mm/yyyy</i>	(e.g. 31/12/1994)										
<i>(ISO) yy-mm-dd</i>	(e.g. 94-12-31)										
<i>(ISO) yyyy-mm-dd</i>	(e.g. 1994-12-31)										
<i>(Julian) yyyy-ddd</i>	(e.g. 1994-365)										
<i>Enumeration</i>	<p>Making a field enumerative means in the application the entry field has a drop-down list out of which the user has to pick an option. If you set this field to <i>Enumeration</i>, you must also fill in the <i>Enumeration values</i> tab.</p>										
<i>European date (dd/mm/yyyy)</i>	Can only contain dates like 31/01/2002.										
<i>HTML</i>	An HTML field is meant for long, laid-out text.										

	<p>Layout can be applied to the text during editing of the record.</p> <p>The user can print the contents of such a field to a Word template or with the aid of an XSLT stylesheet, whilst keeping the layout intact. Although one can only see the laid-out text while editing an HTML field, the field contents will actually be stored as HTML code in the background.</p> <p>You can use the field as an alternative for the older and rather narrowly applicable <i>Rich text</i> field type.</p> <p>Click here for more information about implementing an HTML field: for instance, it is not enough to specify the data type here, you must add or change a screen field as well.</p>
<i>Image</i>	<p>When this field is to contain paths to images, you must assign the <i>Image</i> type to it. The reason is that when you print this field through an adapl or Word template, Adlib must know whether it should print the path or the image to which the path points. When printing a field of this type, the image will be printed and not the path. (For displaying paths and images in a running Adlib application this type setting is not relevant: an image viewer will automatically display an image, and an entry field will display the path.)</p> <p>As soon as you choose this data type for a normal field, the extra tab <i>Image properties</i> is added to the properties tabs to further set up this specific type of field.</p> <p>When you add an image field to a screen, remember to add a Media Viewer box to the screen as well, so that linked images can be displayed automatically in the running application.</p> <p>In the running application, the <i>Find image file</i> button becomes active as soon as the image field has the cursor. This function allows you to search your system or network for the image you wish to link, using a standard Windows Explorer component. You do need a Media Viewer on the screen, for the Explorer component to work.</p>
<i>Integer</i>	<p>Will accept only whole numbers. The numeric value</p>

	may be preceded by a plus or minus sign.
<i>ISBN</i>	Can contain a valid ISBN, either with or without punctuation (e.g. 90-03-90201-1 or 978-90-03-90201-6).
<i>ISO date (yyyy-mm-dd)</i>	Can only contain dates like 2002-01-31.
<i>ISSN</i>	Can contain a valid ISSN, either with or without punctuation (e.g. 0040-9170).
<i>Letters (A-Z/a-z) only</i>	Will only accept alphabetic characters and spaces.
<i>Logical (Boolean)</i>	A logical field is a field which is one position long. If the field contains any character, its contents are considered 'true'. If it does not contain a character, the field contents are considered 'false'. On screen, a normal screen field associated with a data dictionary tag of the <i>Logical</i> type will be represented by a checkbox. When the user marks a checkbox, Adlib saves an 'x' in this field. So, for searching on marked checkboxes, the user should search on the letter 'x'.
<i>Numeric (floating point)</i>	Will only accept numeric characters. Use a full-stop as the decimal symbol. The numeric value may be preceded by a plus or minus sign.
<i>Rich text</i>	A <i>Rich text</i> field contains text that can be laid out by the user. This layout is only visible on screen though. For printing the contents of a <i>Rich text</i> field, the layout is not used: text will be printed plain. Because its limited applicability, this field data type is now deprecated: we recommend you use the <i>HTML</i> field data type instead.
<i>Temporary</i>	Use this type for fields in which you want to store values temporarily (for instance for displaying temporarily compounded field values). Fields of this type always have full rights, which means they can be written to, while the database may only have read rights. Therefore these fields will not be stored in the database and will also not appear in field lists in Adlib applications. Apply temporary fields when writing to the database(s) is not possible or not allowed, for example because they might be kept on cd-rom, or when for security reasons the database has no write access (for some or all users).

	Note that even for just displaying some compounded fields in old Adlib applications (that have not been assigned the <i>Temporary</i> type) it is necessary for the database to have write access for all users. That is because putting together values from other fields only happens just before display (through an adapl), which then has to be written in a database field (!) to make display possible, even although that storage is temporary; see for instance fields in some brief displays, or the BR field (<i>[display_only]</i>) in Library applications in which title and author(s) are put together. These fields are not displayed when the database cannot be written to.
<i>Text</i>	Will accept all characters.
<i>Time</i> (<i>hh:mm:ss</i>)	This field will accept a time. The only accepted notation is <i>hh:mm:ss</i> (e.g. 23:55:00)

Presentation format

From Adlib 6.5.0, the contents of fields of data type *ISO date* or *Numeric* may be presented on screen differently from the way that content has been stored, to allow for automatic language or region-specific format presentation.

- *ISO date presentation*

In Adlib databases, dates can occur in different formats, for instance as European date (dd-mm-yyyy), American date (mm/dd/yyyy) or ISO date (yyyy-mm-dd). This is not beneficial to the international exchangeability of data which contains such dates. That is why in new Adlib applications, date fields in the data dictionary are set to the ISO date format more and more. To still be able to present dates on screens in the locally used format, it is possible to set a fixed or variable presentation type for ISO date fields in the data dictionary, using the current property. You have a choice between the following presentation types:

- *ISO date (yyyy-mm-dd)*: the date format is presented unchanged, this is the default case for existing ISO date fields.
- *European date (dd/mm/yyyy)*: this ISO date field will be presented in the format commonly used in Europe.
- *American date (mm/dd/yyyy)*: this ISO date field will be presented in the format commonly used in America.

- *Locale date (short)*: this makes the presentation of this ISO date field variable, by linking it to the local Windows language. For example, if this language is English then Adlib will automatically present this date field in the European format.
- *Locale date (long)*: this presentation too is variable and linked to the local Windows language. However, the presentation is a date half written in words. For English that could be something like *Tuesday 2 December 2003*, for example.

The storage of an ISO date field will always be in ISO format, regardless of the presentation you choose for the field.

Entry of an ISO date is normally done through a small calendar, from which you pick a day. However, after your choice, the date will be presented in the selected presentation format.

Note that the *Locale date (long)* format needs more space on the screen. So for this presentation you'll have to lengthen the relevant date field on all applicable screens, before you'll be able to properly observe the new presentation in a running application.

Searching on date fields in the *Search wizard* and in search forms, is self-explanatory. In the *Expert search system* you can search in differently presented ISO date fields in two ways, namely on ISO date and on the date in the presentation format. So, an ISO date field which has the European format as presentation format, can be searched equally well with ISO dates as with European dates. An exception to this rule is the ISO date with the *Locale date (long)* presentation format: these fields can be searched on ISO dates and on *Locale date (short)* dates. So you can't search these fields on, for example, "Wednesday 15 April 1970".

- Numerical value presentation

The entry and presentation format of fields of the *Numeric* (floating point) data type can also be linked to the local Windows settings for it. This presentation setting doesn't affect the way a numerical value is stored: that is specified through the *Decimal separator* option for the database. The numerical value 112.50 will be presented or can be entered as 112,50 in a Dutch Windows version, whilst that would be 112.50 on English Windows systems.

From 6.6.0, the local presentation format also ensures that dots (in Dutch) or comma's (in English) to separate factors of thousand, like in 1,000,000.50, are added to the presentation of a number (but not stored in the record). You can enter a number with or without the locally valid separator character; if you do that without the separator, then Adlib will add it to the presentation as soon as you leave the field. For example, if you enter the number 123498.90 in such a field, then Adlib will change this number to 123,498.90 when you leave the field.

The presentation options available are:

- *Default*: values must be entered in the format set by the *Decimal separator* option for the database, and are presented that way as well.
- *Locale (numeric)*: present the value in this field according to the local Windows settings.

Member of group

If you want a data dictionary field to be part of a group of fields in the data dictionary, you should give all these fields the same group name. A group of fields is used to repeat or move together when you add or delete a group occurrence for one of these fields in a running application.

You can define a group name here, on data dictionary level in the database setup, and/or a group number on screen level in the application setup.

When you want groups of fields on a screen also to be accessible as a group from within ADAPL or Adlib Internet Server, you should define the group in the database setup only, although setting up the same group on screen level too, won't cause conflicts. Data dictionary groups do have priority over screen level groups.

It's possible to define groups only on a screen, but their use is limited to the application that incorporates that screen. Adlib advises to only use data dictionary groups.

Note that certain ADAPL functions (like `NULL`, `REPSORT`, `REPINS` and `REPCOPY`) have a different effect depending on whether a field has been defined in a data dictionary field group or as a screen field group. So replacing an existing screen field group (usually in older applications) by a data dictionary field group definition, may have relevance for adapls using the above mentioned functions. If a field has to be in a field group, it is recommended to define the field group in the data dictionary (and only in the data dictionary).

Further note that data dictionary field group functionality like adding or sorting group occurrences, with the exception of `REPCNT`, is switched off during import of data. Any adapls used for importing should take this into account when manipulating data.

Exchangeable

With this option you determine whether the contents of this field will be copied to a new record when you copy an entire record in Adlib or when you derive a record with this field. Mark this checkbox to allow copying of the field.

Note that before Adlib 5.0 an adapl (a copy record procedure) was needed to prevent specific fields from being copied, when copying an

entire record.

Occurrence sort order

With this option you determine whether the occurrences of this field will be sorted or not, before the record is saved. You can choose between *Do not sort* e.g. (z, a, c) or (2, 5, 1), *Sort ascending* (a, c, z) or (1, 2, 5), or *Sort descending* (z, c, a) or (5, 2, 1). The type of the sorting (alphabetical, numerical or by date) depends on the data type of the current field; alpha-numerical sorting of occurrences, as would be handy for text fields which also contain numbers, is not possible. If this field occurs in a data dictionary group, then that group stays together after sorting, as is desirable. But if this field does not occur in a data dictionary group but does occur in a screen group (the group is only defined on interface level), then you'll have to create a data dictionary group of this group before you may apply sorting to a field in it; if you apply this option to groups that are only defined for the screen, then the concerning field is sorted but the accompanying group data isn't, so screen group occurrences become chaos! In many existing Adlib applications, groups are only defined on screen level, so check your data dictionary before applying *Sort ascending* or *Sort descending*.

If a sort is applied to more than one field in a (data dictionary) group (which is not permitted), then the sorting order is determined by the last sort field in that group.

In the properties of an internal link definition you may also set the *Sort tags* for occurrences of fields. This may result in conflicting sort instructions if you set both sort options differently for internally linked fields. But Adlib simply first executes any sorting set up in the internal link definition and after that, any sorting set up in the field properties.

Multi-lingual

This option only applies to SQL and Oracle databases. If you make a field multilingual this way, the user can enter a value for this field (and each occurrence of it) in any language that you have set in the *Data languages* list for the .pbk file of this application. In a running Adlib application on such a database, the user must choose the language in which he or she wishes to enter data in this field, from the *Data language* button submenu (this menu only offers the languages set in the *Data languages* list).

If you want to create multilingual fields, it's best if those fields do not contain any data yet. This is because when a field which already contains data is made multilingual, the data in the field will not get a language attribute automatically. However, from Adlib 6.6.0, when a

record with a multilingual field is written again (when edited and saved manually), Adlib will check if the field data already has a data language attribute and if not, it will add the current data language as the attribute. To be more precise: if in a record multilingual fields occur in which data is present without any associated data language, then on retrieval or display of the record – it doesn't need to be in edit mode – the currently active data language will be assigned to the relevant field values in memory, but not stored in the record yet. You can observe this preliminary language assignment while viewing the current contents of the record using the **Ctrl+R** key combination. If the record is put in edit mode, the following message will appear: *Monolingual data detected and promoted*. You can still edit the translation(s) of the field values of course, by switching data language or via the *Edit multilingual texts* window. Save the record to store the changes; if you do not save the record, all remains the same and the field values without associated data language will still be present.

In the XML format in which every record is stored in the aforementioned database types, every translation of a value in a field occurrence is saved in its own XML tag specific for the relevant language: this tag is the code for the language as shown in the *Name* column in the *Data languages* list in the application properties.

Link reference tags must never be multilingual, even if they are associated with a multilingual linked field.

After making a field multilingual, you must reindex the index on the field (only if it already has an index). If the field is a long text field you must reindex all word indexes, to rebuild the wordlist as well.

(Note that there are two ways of implementing multilingual fields; [click here](#) for more information.)

Do not show in lists

Apart from fields of the *Temporary* data type, all fields which have been defined in the data dictionary will be shown in field lists such as those in the *Expert search system*, the *Sort* window and the *Replace...* window.

Still, it may be that you wish that certain fields (of other data types) wouldn't be in those lists, like for instance the fields with the tags *FD*, *FF* and *FG* in the *DOCUMENT* database, which can contain the name of a dataset in which the user has written a new catalogue title record from one dataset to a manually chosen dataset. There is no point in including these fields in the field lists, and may even be confusing. Mark the *Do not show in lists* option to keep the currently edited data dictionary field from the field lists in your Adlib application (available from Adlib 6.3.0).

Exclude from full text index

<no help available yet>

4.11.7.2 Linked field

4.11.7.2.1 Linked field properties

On the *Linked field properties* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Linked field* on the *Field properties* tab, you set up the link to a field in another database.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Folder

This property contains the full or relative path to the database you want to link to, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the relative path to that folder is automatically entered here. Usually, you keep your databases in one folder, and the relative path you'll often encounter here is: `../data`. A single dot represents the current folder (which is the case for internally linked fields).

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you want to link to. Do not enter the extension of the file. Examples of database names are `DOCUMENT`, `COPIES`, and `Collect`. A single "=" symbol in the *Database* property represents either the current database or the current dataset (which applies to internally linked fields), depending on the *Database scope* setting for the relevant internal link definition. If the *Database scope* property has been set to *Undefined* or *Database*, "=" refers to the current database, while if the *Database scope* property has been set to *Dataset*, "=" refers to the current dataset (within the current database).

Important: when you work in a copy of your live application, then make sure you search the right folder (in the copy) for the proper file: otherwise the relative path will be incorrect when you place back this copy as your live application later on.

If the database that you select has datasets defined for it, these

datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) Selecting a dataset is optional, you can also just link to an entire database. Typically, you select a dataset if for this link you only want to retrieve data from that specific dataset, and if you want new linked records to be created in that dataset automatically. If you select no dataset, when the database does have datasets, you must offer the user the choice of dataset when he or she creates a linked record from within the current primary database (see the *Dataset selection field* option at the bottom of this tab).

Lookup field

Enter the tag or the field name from the linked database in which Adlib must search for the key data when accessing the currently defined field in a running application. A term (!) index must have been defined for this lookup field in the linked database. (If you want to use a free-text field, such as title, as lookup field, you should create an (additional) index on term for that field. Then make sure the index *Key size* option is set sufficiently long, although maximally 100 characters for indexes in SQL databases, while indexes for CBF database can be longer, e.g. 255 characters, and increase the index *Page size* as well, so that a title can be indexed as a single term in this index. Also, if a free-text index already exists for the field, then the new term index cannot list the tag to be indexed as the first tag, so a dummy tag must be listed first and secondly the free-text field.)

The best way to select a lookup field, is when you have selected a database to link to first, then to click the ... button next to this option to open the *Database fields* window, and just pick the desired field from the available fields in the linked database. The presented fields list is limited to fields which actually have a term index. This way you can't accidentally select a field with a different type of index as the lookup field.

Note that fields do not have to be text fields per se, to have a term index. The *Type* column in the *Database fields* window indicates the field type, not the index type.

Further note that fields in the linked database which do have a term index in there but haven't been defined in the data dictionary of that database (which is a rare situation), do not appear in the list of lookup fields. Then still define the field in the linked database to solve the problem.

Forward reference (a.k.a. link reference)

Enter the unique tag or field name of the field in the primary database in which Adlib must save the record number of the linked

record. This is necessary if you want modifications in the key data in the linked record to be visible in all the records that refer to it. If you specify a link reference field, Adlib will not save the key value of the linked field in the primary database, only the record number that points to the linked record.

After you've entered a tag or field name, and you leave this property, Designer will check if the tag or field name is not in use as a forward reference by another linked field already, and warns you if it is used elsewhere; then choose another tag or field name.

If you do not enter a link reference field here, Adlib will allow the old link to expire as soon as a user modifies the key data in the linked record: the linked fields in the primary database keep the old values. This might lead to incorrect values remaining in the primary database.

If you use the linking method with the link reference field, the key data field in the primary database cannot be indexed. This is because the data for the key field is not saved in this database, only retrieved temporarily. Instead, you can and should* define an index (of the integer type) for the link reference field. (The only place where you refer to this indexed link reference tag instead of the linked field itself, is when you create a method/access point for the current linked field. In screens you always associate an entry field with the linked field name or tag, never the link reference tag.)

If you use this link reference in a reverse link setup or a feedback database, it is required to create an index of this link reference field.

In existing Adlib applications not all forward references are specified as data dictionary fields too. This is because specifying a forward reference tag here, is enough to create the tag in the database, with the right properties. (This allows working with forward reference tags in ADAPL too.) But this practice has proven to have its disadvantages: there is a risk of using non-unique tags because the implicitly defined forward reference tags do not appear in the data dictionary and derivative lists of fields in the current database. Therefore, Adlib now recommends to create a field definition in the data dictionary for each forward reference. Specify the *Data type* of a forward reference field as *Integer*, and set *Member of group* to the group name to which the associated linked field may belong. And if the associated linked field is repeatable, then also make the forward reference field repeatable. The forward reference field is now a part of the data dictionary, is well documented, and will appear in all field lists in Designer and running applications. Now, when you select its tag or name for the *Forward reference* option in the *Linked field properties*, you can be sure the tag is unique.

It is not necessary though, to create the forward reference field in the data dictionary prior to assigning its (future) tag or field name to the current property, because when you enter a non-existing tag here, Designer automatically adds a data dictionary field definition for

this tag in the current database and sets the *Data type* to *Integer*, copies any group name and *Repeatable* setting. But do check that the settings are right, also after you change the definition of the linked field.

* You should always make an index for a link reference field. Strictly speaking it isn't always a requirement, but it's good practice to make it a rule.

Backward reference (a.k.a. reverse link)

Enter or select the forward reference tag used in the linked database to link to the field in the current database, that you are specifying now. The field in the linked database that is associated to that forward reference, should be reciprocally defined. (See Reverse links for more information.)

Fixed domain

Select *Fixed* if you want if you want this field to always retrieve its data from the same domain in the linked database. Also fill in the *Name of the domain to link to* option; not filling in a domain name results in a corrupted index.

Variable domain

Select *Variable* if you want to allow the user to choose a domain each time he or she enters a term in this linked field. Also fill in the *Tag which holds the domain name to link to* option.

Name of the domain to link to

Only for a *Fixed* domain. fill in the name of the domain in the linked database from which this field must retrieve its data. This typically applies to fields in databases (like the catalogue) from which you link to authority databases (like the thesaurus). The specified fixed domain must of course exist in the concerning validation database.

Leave this option empty if you don't want to link this field to a domain.

Tag which holds the domain name to link to

Only for a *Variable* domain, fill in the tag or field name in the current file (not the linked file) in which the applicable domain names are saved.

This typically applies to fields in databases (like the catalogue) from which you link to authority databases (like the thesaurus). Said

domain field, in for instance a catalogue, should preferably be an enumerated static list from which the user can choose a domain for the currently specified linked field. So in an application you'll then have two entry fields (usually above each other), of which in the first the user has to fill in some term, and in the second select a domain to which that term should belong. When setting this up in the current primary database, you should fill the enumerated list with the appropriate domain names yourself.

Strict validation

This checkbox should always be marked. The option to deselect it is obsolete and may lead to data corruption if the *Allow creation of new linked records* option below has been deselected as well. To be precise:

- if both this option and *Allow creation of new linked records* have been deselected, then any entered term in the linked field will be stored directly in the linked field itself. If the current field is linked on reference, this leads to data corruption since the entered value is stored in the local record instead of in a linked record, while if the current field is not linked on reference, this leads to a discrepancy between terms registered in the linked database and terms registered in local records, which is undesirable as well.
- if this option has been deselected while the *Allow creation of new linked records* option has been marked, a new linked record will be created automatically for any new term entered in the linked field, but the user won't be notified of this in any way. This is undesirable because instead of suggesting the user to select an existing term if possible (and thereby keeping your authority files as clean as possible) the user may incorrectly assume that any entered term is validated and therefore be unaware of redundant, non-preferred terms being entered.
- Mark this option to make sure that any entered value in the linked field is validated against the linked database. If the *Allow creation of new linked records* option has been deselected, the user will be forced to pick an existing term from the *Find data for the field link window* before leaving the linked field, while if the *Allow creation of new linked records* option has been marked, the user can choose between picking an existing term from said window or adding the new term as a new record in the linked database explicitly.

Link only first occurrence

As a rule of thumb, always mark this checkbox. It ensures that of any merged-in fields only the first occurrence will be retrieved. This is

essential if the current linked field is a repeated field, which is very common, because Adlib cannot deal with repeated fields within a single repeatable field group occurrence. Note that any (repeatable) data dictionary field group may contain more than one linked field that links only to the first occurrence.

You are only allowed to deselect this checkbox if the current linked field is not a repeated field and if the fields to be merged in are not part of a data dictionary field group: single repeatable source fields to merge in are allowed though. You could then add individual repeated target fields in the main database and on screen, to contain the multiple occurrences of the repeated fields merged in with the current linked field.

Allow creation of new linked records

Mark this checkbox to allow the Adlib user to add or force (adding and forcing are two ways of creating new linked records) new records into the linked database, by entering a new term or name in the linked field and explicitly add or force the new record from within the *Find data for the field* link window.

If you deselect this checkbox, the Adlib user will not be able to add or force records to the linked database from the current linked field and he or she will always have to pick an existing term or name from the linked database through the *Find data for the field* link window (provided the *Strict validation* option above has been marked). Nor will the Adlib user be able to edit any existing records in the linked database, through the current linked field.

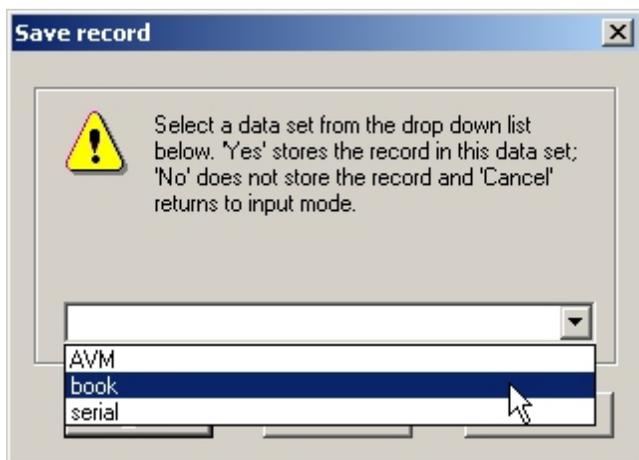
Destination dataset field

If you allow the user to create new records in the linked database from within the current primary database, and this database is divided into datasets, you have choose whether you want the new linked records always to be created in the same dataset or in a dataset that the user chooses each time a new linked record is to be saved.

If you retrieve the data for this linked field from a fixed dataset, that you set in the *Dataset* option in the top part of this tab, you automatically save new linked records in that dataset too.

If you want the user to have the option to choose a dataset, then you leave the *Dataset* option in the top part of this tab empty (to be able to retrieve data from the entire database), and you provide a *Destination dataset field* instead.

In this option you enter the tag or field name that will contain the name of a specific dataset to which a new linked record must to be added. The user chooses a dataset from a static enumerated list in an automatically created dialog in the running application when the *Edit screen* for the new linked record is closed.



For example, in the *orders* database (Acquisitions module) there is a linked field *ti* to the *document* database. In the running application this is the entry field *Title*. The user may choose a title from the entire catalogue to order it, so in the *Linked field properties* for *ti*, no *Dataset* is specified. If the user is allowed to create a new linked record, for instance when an order is entered for a title not yet registered in the catalogue, the user must be able to specify whether the new title is a book, a serial, an audio-visual material, etc. So in the data dictionary another field has been created with the tag *FE*. This is an enumerated field with a static list that includes *book*, *AVM*, and *serial*. For the tag *ti*, *FE* is specified as the *Dataset selection field*, and that's all.

Note that you mustn't use the *Destination dataset field* functionality for internally linked fields. Although the Adlib software will display the dataset selection window to the user, the new linked record will not be created in the dataset chosen by the user, but in the database as if it wasn't divided up in datasets.

External sources

It is possible to use multiple (external) thesauri in Adlib, when searching or validating a linked field set up for it. In the *Search wizard* and in the *Find data for the field...* window (*Linked record search screen*) the user can then find the *Thesaurus* drop-down list in which he or she may choose another thesaurus and select it for use.

There are no preset external sources by default. From Adlib 6.5.0, any extra thesauri which might be used, must be set for the applicable linked field(s) first, here in Adlib Designer. The *Thesaurus* drop-down list will only become visible if the user searches the relevant linked field. The user may then choose between the external authority files

preset by the application manager only.

You set external thesauri for the currently selected linked field by adding new data sources to the *External sources* list:

1. Click the *Add* button to add a new data source.
2. Put the cursor in the *Path or URL* entry field and click the ... button to look for a thesaurus *.inf* file on your system or on the local network. The path to the located *.inf* will be filled in for you. You may also enter a path manually or provide a URL to a web service which can search a database on the internet for a key entered by the user to receive the search result back in Adlib XML format. For example, the Dutch version of the AAT (Art and Architecture Thesaurus) is available online and accessible via: `http://service.aat-ned.nl/api/wwwopac.ashx?database=aat-gateway&search=term[nl-NL]="%data%*" sort term[nl-NL]&limit=100` This URL allows truncated searching in the term field of AAT records. The [nl-NL] parameter is required because even though this AAT service yields only data in the Dutch language, the term index used for searching is multilingual. The double quotes around %data%* are required to handle spaces in search values properly. Further, the limit can be adjusted to suit your needs. Set the limit appropriately high to compensate for the fact that the limit is the maximum number of keys returned and you cannot page to a next batch of keys, but also note that the higher the value, the longer a search may take.
3. Click the *Description* entry field to be able to enter the name or description of the external thesaurus in multiple languages. This name will become visible in the *Thesaurus* drop-down list in Adlib. The official description for the Dutch AAT is: AAT-Ned, Copyright © RKD & Getty
4. Click the *Link screen* entry field to select a link screen for this external source. You may have to create a link screen especially for this external source. The link screen must contain a small number of Adlib field tags (say 1, 2 or 3 tags) in a row with the associated labels above (!) each field. So a link screen contains only two lines of screen elements: the top line consists of labels for the field tags below them in the second line. Adlib doesn't display this screen like other normal record screens in an application: the link screen is converted to the *View table* tab in the *Find data for the field...* window: with the labels on the link screen you specify the column headers, and with the field tags the contents of the columns on that tab. The tags that you place on the link screen must be Adlib field tags from the linked database, whose field names match the field names from the XML returned by the external source; if the external source returns

different field names, then the XML must be transformed by a local web service using an XSLT stylesheet before the Adlib software takes control over the XML output.

The online AAT provides the following field names in the search result: *aatNedTermId*, *term*, *broader_term*, *narrower_term*, *scope_notes*, *term.code*, *term.status*, *source*. Of these fields, the following are present in an Adlib 4.2 model application (the associated tags are mentioned as well): *term* (te), *broader_term* (bt), *narrower_term* (nt), *term.code* (tc), *term.status* (ts) and *source* (br).

Whenever the user searches for a term in this external source, the *View table* tab offers the user a list of terms to choose from.

5. If you wish, you can provide more external thesauri for this linked field by repeating these steps.

The *Advanced* button (available from Adlib 7.1.0) opens the *External source: advanced properties* window, which allows you to set advanced options for the currently selected external source. This functionality (sometimes called the annotation tool) includes the ability to view and retrieve more information from the source record than just the term or name, like source information and scope notes or any other descriptive data (as long as that is present in the source database and it is mapped to one or more fields in the local authority database, using these properties), to allow you to pick the proper term from the term list in the *Find data for the field* window more easily. It also offers grouping and sorting options for the retrieved term list, to be able to choose a term or name from a particular source for example (when term or name records have been registered as coming from different sources) or have the list of terms sorted in a particular way. To this end, using the advanced properties discussed here, the *View table* tab in the *Find data for the field* window can be set up to display multiple lines of data per found term or name and is also able to display headers above collapsible and expandable term/name groupings. The *External source: advanced properties* window offers the following options:

- **Adapl** - Add a reference (relative path) to an adapl by (typing it or by using the ... button), which formats each term or name and its accompanying data for a multi-line presentation in the term list on the *View table* tab of the *Find data for the field* window in your local application. Firstly, to display the *View table* tab at all, a link screen needs to have been specified on the *Link screens* properties tab of the current linked field, but all the fields and labels on the selected link screen will be overruled by the adapl you specify here. The relevant adapl will be executed per listed term or name record and has access to all data from the XML document resulting from the query at the basis of this external

source, provided that the local, English data dictionary field name of each field tag you use in the adapl is identical to the field name in the XML search result.

In the adapl, the data from the field tags must then be assigned to different occurrences of two special reserved variables named `&X` and `&Y`. Each occurrence of `&X` represents a data line on the *View table* tab in the *Find data for the field* window, where window, where the smallest occurrence number is the top line of the relevant term data and the biggest occurrence number the bottom line. These occurrences can be assigned strings (of fixed text) and/or the values from variables or field tags, whereby values and strings can be concatenated using the `+` sign like when building strings in output adapls. Further note that the occurrence numbers do not need to be consecutive to have the same result, so specifying occurrences 1, 2 and 3 has the same result as specifying occurrences 4, 10 and 30 (if you wouldn't find the latter confusing).

Each occurrence of `&Y` represents the font colour of the text displayed through `&X`. These occurrences must be assigned an HTML hexadecimal colour code as a string value and there must be as many occurrences of `&Y` as there are of `&X` and their occurrence numbers need to be identical so that e.g. colour occurrence 4 can be matched to data occurrence 4, etc. You can display as many data lines per term or name as your wish. The full ADAPL code example below displays three data lines per term, containing the term itself in line 1, the scope note in line 2 and source in line 3, each text line in a different colour:

```
&X[1]= te;
&X[2]= sn;
&X[3]= 'Source <' + br + '>';
&Y[1]= '#000000';
&Y[2]= '#303090';
&Y[3]= '#A0A0A0';
end
```

- **Group by field** (optional) - Use the ... button to select the field which you'd like to use to group the search result before sorting it. Typically you would want to use an enumerative field (of similar field with a limited number of values) for the grouping, never a field with unique values. Each group value will be displayed as a bold header above the terms or names that are part of the relevant group. In the adapl specified above, you don't need to program anything for the group-by field. Note that setting this option means that the grouping will be handled by the Adlib client, *after* the search result has been retrieved from the server.

- **Initial group size** - Specify the maximum number of terms or names that should be visible per grouping when the list of terms is displayed initially. This number doesn't limit the actual search result: by clicking the hook icon on the right side of the group header label in the terms list you can expand a group to display all the terms underneath it. A box icon means that all terms are displayed already. Setting this value to zero means that all groups will initially be displayed collapsed.
- **Sort options** (optional) - You can sort the search result, *after* the search result has been retrieved from the server, on one or more tags. Click the *Add* button to add a new sorting line. Then click the *Sort tag* entry field to select the field tag to sort on, select the *Sort order*, and if you always want to sort a particular field value on top, then enter that text value in the *Sort priority* entry field. The sorting conditions you set here, will be applied from top to bottom.
- **Mapping** (optional) - Even though you might use these advanced properties to display other fields from the source term record as well as the term itself, that doesn't mean the other data is copied to the local record when you use the relevant term from the external source: normally only the term or name is copied. To map a field, click the *Add* button to add a new mapping line. Then click the *Tag* entry field to select the field tag to map. Although it is possible to map many fields, we recommend to map no more than the source and URI data, and take into account that mapping internally linked fields can possibly create unforeseen and/or conflicting local relations between terms. One (and only one) of the mapped fields can be marked as the *Primary key*, and this field will then be used to find out if the external term selected by the user already exists in the local authority database or not: if you select such a field, it must be field in which all values are unique, like a URI field for example (the term field can often have non-unique values, but is used by default if you specify no primary key field here). And if the term or name cannot be found in the local authority database as a value in the field set as primary key, a search in the term/name field will be performed instead.

(See the Adlib User Guide for more information about the external sources functionality from a user's point of view.)

4.11.7.2.2 *Relation fields*

On the *Relation fields* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Linked field* on the *Field properties* tab, you may also link the current linked field to the thesaurus tags related hierarchically to the field you link to. (You need to have a thesaurus to apply this functionality.) Specify one or more of these tags or field names if you want those relations to enrich the list of terms that is displayed in the *Linked record search screen* (which columns you define in link screens), when you search for existing terms in this linked field.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Preferred field

If a tag is filled in for *Preferred field*, Adlib will check for the terms resulting from the search whether a value occurs in their preferred field tag. If so, the term will be a non-preferred term, and a stop sign will appear in the *Find data for the field...* window (a.k.a. the *Linked record search screen*) for that term.

Equivalent field

If for the terms resulting from the search equivalents are available in the tag provided here, these will be listed in the *Find data for the field...* window too.

Broader field

If you fill in a tag for the *Broader field*, then if a broader term is available for the terms resulting from the search, those terms will be displayed hierarchically in the *Find data for the field...* window.

Narrower field,

If you fill in a tag for the *Narrower field*, then if a narrower term is available for the terms resulting from the search, those terms will be displayed hierarchically in the *Find data for the field...* window.

Semantic factor field

If the terms resulting from the search, are a semantic factor of some more complex term (the concept term), then the more complex term will also be included in the list in the *Find data for the field...* window. Specify the tag of the *Semantic factor of field* in the thesaurus (if present).

Range start field

This property is meant for period name linked fields which are being used to implement a NamedRangeMethod. In that case, enter here the field name or tag of the field in the linked database which contains the start date of the period of which the name is defined there.

Range end field

This property is meant for period name linked fields which are being used to implement a NamedRangeMethod. In that case, enter here the field name or tag of the field in the linked database which contains the end date of the period of which the name is defined there.

Context field

Use this option if you want to show the upwards hierarchy of a linked term in a special, read-only field for this purpose, on some detail screen. Enter the tag of a text or temporary field from the current database which will be filled by Adlib with the hierarchical context of the linked term, yet only its broader terms and the term itself, in the following syntax: *top_term/bt/bt/.../bt/term*, if the currently specified linked field links to a field in another database which has broader and narrower term fields. See the Showing the upwards hierarchy of a linked term topic for more information.

See also

Internal links

The Adlib User Guide (for information about hierarchical relations in the *Thesaurus* and *Persons and institutions*)

4.11.7.2.3 *Link screens*

On the *Link screens* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Linked field* on the *Field properties* tab, you may specify search and zoom screens to access the linked database from a linked field in the current database. Zoom and edit screens may instead also be specified in the entry field properties on a screen; this allows you to set different linked zoom/edit screens for different "primary" screens. But usually you want to use the same linked screens for every "primary" screen that contains this linked field. So specifying your linked screens in the data dictionary saves you work in the

application setup.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Link search screen

Select the screen to be used as a *Query by form* for searching in the linked file from the currently specified linked field in the primary database. Such a *Query by form* appears when the user has entered this linked field, and clicks the *List* button (**Shift+F4**). Preferably, use relative paths like `../screens/qbdoc`, and name the screen without extension. A search screen is of course optional.

Typically, you want to offer the user a *Query by form* if the linked field only contains a number which is the key to the linked records (like reproduction *Reference*); in the *Query by form* the user can usually enter several criteria to search for the needed number, in the example of the reproduction *Reference* these criteria could be fields like *Creator*, *Technique*, *Location*, *Title*, etc. If the search performed from such a *Query by form* yields more than one result, these records will be listed in a *Linked record search screen*: this is either a standard link screen for the current linked field, or the *Link screen* specified in the option below. The user makes the final choice from this list.

Link screen

Select the screen which Adlib must use to display a list of possible values for the linked field on the *View table* tab in the *Find data for the field...* window (better known as the *Linked record search screen*), when the user presses **Shift+F4** or clicks the *List* button. Note that the *View table* tab won't be present for this field if you do not specify a link screen here.

A link screen contains only two lines of screen elements. The top line consists of labels for the fields below them in the second line. Adlib doesn't display this screen like other normal record screens in an application: the link screen is converted to the *View table* tab in the *Linked record search screen*: with the labels on the link screen you specify the column headers, and with the fields the contents of the columns on that tab.

If you also specify a *Search screen* in the option above, then that *Query by form* will be displayed to the user first when he or she wants to search this linked field (**Shift+F4**). If the search performed from such a *Query by form* yields more than one result, these records will be listed in the *Linked record search screen*, also by means of the link screen you specify here.

If you specify no link screen in this option, Adlib will display the standard *Linked record search screen* automatically, when the user

enters a value in the linked field that points to more than one linked record, or when a search through a *Query by form* yields more than one record. This standard *Linked record search screen* contains one column of values for the currently linked field, in the form of a tree structure on the *View hierarchy* tab, plus a *View lists* tab containing pointer files. How detailed the hierarchical tree structure on the first tab is, depends on the *Relation fields* that you have specified for this linked field.

The fields that you place on the link screen must be fields from the linked-to database; for this purpose you don't need to define those fields in the primary database as well.

[Click here](#) for more information about designing different types of screens for use in Adlib.

Do not show link screen

Mark this option to display all available terms from the linked database (often from a specific domain as well) in a drop-down list instead of in the *Find data for the field...* window (aka the "link screen").

This functionality is available from Adlib 6.6.0. It allows you to offer user-friendly drop-down lists without the advanced options of the *Find data for the field...* window, and the relevant terms can still be edited via *Make/edit linked record* or via the thesaurus. If the field is empty (no value from the drop-down list is currently selected) then you can add a new term via *Make/edit linked record* too. Adding new terms for this field can be done in the thesaurus as well. Since long drop-down lists are not user-friendly either, we recommend to use this functionality only if you'll know in advance that the terms list will be limited to a small number of terms. Long lists are still possible though; if a list doesn't fit on the screen, a scroll bar will appear with which you can scroll through the list.

Link screen Adapl

Select the name of an ADAPL procedure with which you can control the list of records displayed in the above specified link screen.

You can use `SELECT NO` (followed by `END`) in this adapl, to exclude certain retrieved records from the list displayed in the *Linked record search screen*.

If no filtering is needed, you don't have to specify any link screen adapl.

Zoom screen

Select the screen in which Adlib must display some selected details of

a linked record from a secondary data source, when the user clicks the currently specified underlined linked field in a displayed record in the primary data source, or when the user clicks either the *Create record* button or the *Details* button for a selected record in the list in the **Shift+F4** *Find data for the field ...* window (*Linked record search screen*) for the currently specified linked field. The zoom screen will be opened in a separate window.

There is no default zoom screen for these situations, so the user cannot 'zoom' in a linked field without a zoom screen specified for it. In this case, a linked field value will not be underlined.

You can also specify a zoom screen for a screen field in the application setup; such a zoom screen has priority over any zoom screen specified here in the database setup.

Whether or not the user can edit the record displayed in this zoom screen depends on two things: first, if the zoom screen is read-only (this can be set in the properties of the screen), no one can edit the record in this zoom screen; secondly, if the zoom screen is editable (not read-only) and in the option below no edit screen has been specified, then users can edit the linked record via this zoom screen.

Before Adlib 5.0.3., the zoom screen you set here, was also used to display details of a linked record from the same primary data source (internally linked records), like for the hierarchical relations in the Thesaurus; if you open the detailed presentation of a term, then any broader, narrower and preferred terms and the like are displayed underlined, and if you clicked one of these, a zoom screen opened. From 5.0.3 however, when you click an internal link, you open the linked record in full detailed presentation in the current Adlib window, without making use of any zoom screen that may or may not has been set (the zoom screen is ignored). This way the user can browse between the detailed displays of all internally linked records that he or she comes across.

The fields that you place on the zoom screen must be fields from the linked-to database; for this purpose you don't need to define those fields in the primary database as well.

Zoom/Edit screen

Select the screen in which Adlib allows details of a linked record from another data source to be changed when the user opens a zoom screen directly in edit mode: this happens either when the main record is in edit mode already, and the user chooses to *Make/edit linked record* via a button or when the user clicks the *Create record* button in the **Shift+F4** *Find data for the field ...* window (*Linked record search screen*) for the currently specified linked field. In all other cases, where a zoom screen is opened in display mode first and possibly switched to edit mode subsequently, the zoom screen

specified in the option above is used. This edit screen may be the same screen as the *Zoom screen*, and in existing Adlib applications it often is. An edit screen must of course never be set to read-only, so if you want to set the *Zoom screen* to read-only, you'll have to use two different screens. There is no default edit screen. If you do not specify an edit screen, then users can still edit data from a zoom screen if you specified an editable zoom screen in the option above. You can also specify an edit screen for a screen field in the application setup; such an edit screen has priority over any edit screen specified here in the database setup.

The fields that you place on the edit screen must be fields from the linked-to database; for this purpose you don't need to define those fields in the primary database as well.

4.11.7.2.4 *Linked field mapping*

On the *Linked field mapping* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Linked field* on the *Field properties* tab, you set up the field mapping for fields that you want to retrieve (merge) from a linked record and display on a detail screen in the record in the primary database, and the mapping for fields that you want to write back to the linked database after editing them in the primary database. Note that you do not have to specify a field mapping for the fields you wish to display in any zoom/edit or link screen associated with the linked field in the primary database; this is because when you "zoom" in to a linked record, Adlib really opens the linked database and allows normal access to all fields in it.

[Click here](#) for information on how to edit properties in general. On the current tab you'll find the following settings:

Copy fields from linked record

When you allow a user to select a value in a linked field, you sometimes want other data from that linked record to be displayed in screen fields in the primary database. Take for instance the reproduction *Reference* entry field (*reproduction.reference* in the data dictionary): when a user selects such a reference in the catalogue, you want other data from that reproduction record to be displayed next to the *Reference* entry field, like the path to the digital image, the format of the image and its reproduction type.

For each piece of data to be retrieved, you must specify which (source) fields in the linked dataset are to be retrieved, and in which (destination) screen fields in the primary database Adlib must place

this information for display of the record; and if the linked field has a forward reference you do not need to define the destination fields in the data dictionary of the primary database.

This is not as straight forward as it seems. There are important differences between fields linked on term and fields linked on reference:

- If the currently specified linked field has no forward reference, the data in the linked field, and in any merged fields from the linked record is only retrieved when you fill or edit the linked field. And then this data is saved in the primary record; in this case, destination fields do need to be defined in the data dictionary of the primary database as well.

This may result in several problems. For instance, when you edit the linked record from within the linked database, the data in the primary record remains unchanged (but this is normal for linked fields without forward reference), and the zoom link to the linked record becomes erroneous if you have changed the key field in there. Or when you edit one of the retrieved fields (not the linked field itself) in the primary record, this change will not have any effect on the linked record (when you haven't specified this field as a write-back field too), which causes differences between the primary and the linked data.

- If the currently specified linked field has a forward reference, the data in the merged fields from the linked record is retrieved each time you display a screen with these fields. And the retrieved data is **not** saved in the primary record, it is only displayed. Also, the linked field itself must be included in this merge-field list too, otherwise no value will be displayed in the linked field! So for linked fields with a forward reference, **always** include the linked-to field and the linked field as the source and destination fields in this copy-fields list.

This construction has a potential problem too: if you try to edit one of the retrieved fields in the primary database and you save the record, the old value will reappear (when you haven't specified this field as a write-back field too).

Nevertheless, Adlib recommends to always use forward references for linked fields.

If one or more of the fields to be merged in with a linked field, are linked fields to some other database themselves (let's call them secondary linked fields), then it's often best (although only really required if the merged-in values must be written back to the linked record too, or if possibly more than one occurrence of the secondary linked field must be retrieved) to merge in the link reference fields associated with those secondary linked fields instead of the secondary linked fields themselves; the target

fields in the primary database must then of course also be link reference fields for linked fields. If you merge in a secondary linked field itself, for instance because it won't be editable in the main record, then the target field can be a plain field of the same data type as the merged-in field, but take into account that in this case only the first occurrence of the secondary linked field can be retrieved. If the secondary linked fields are not linked on reference, then you have no choice but to use the linked fields themselves.

If one or more of the fields to be merged in with a linked field, are already merged-in fields (with some other, secondary linked field) in the secondary database, then there's no need to merge them into the current primary database if you already merge the relevant (link reference field of the) secondary linked field into an identically defined field in the primary database (including its own merge list).

Note that merged-in fields are problematic to index, although it is possible to create an access point for them.

Copy lists from linked record

With this functionality you are able to display all occurrences of some source field from a secondary database, in a drop-down list (a dynamic enumerative list) of a destination field in a primary database, dependent on the value in an associated linked field.

This type of enumerative list has not been applied in our model applications yet, so an example may clarify its purpose.

Say we have a primary database in which we document conservation activities in a museum. A part of the data in such a record must describe the environmental requirements for the activity. Some of the necessary fields are the type of *Measure*, the allowed *Value* (range) of this measure, and the *Unit* of measure: the type of measure could for instance be *temperature* or *humidity*, for *temperature* the *Value* could be any single degree or range of degrees, and the *Unit* could then be *Fahrenheit* or *Celsius* for example. In this case we would further like to store the types of measure as Thesaurus records (in their own domain, e.g. *Environment*), and we want each measure record to hold all possible values and units for it, since we want to limit the number of acceptable values and units strictly.

A logical way to implement this, would then be to add two repeatable fields to the thesaurus, one for the units and one for values; the measure type will be stored in the *Term* field. (Of course you must also adjust or create a screen and include it in your application, to display this data and allow user input.) A part of such a filled in Thesaurus record could look like the following:

Term	temperature
Merge list unit	Degrees Celcius
Merge list value	17-23
	15-25

So with this (rather limited) record, the user will have no choice of measure unit, because it can only be *Degrees Celsius*, but there is a choice of two temperature ranges for the conservation activity.

In the conservation activities database we must now create three analogous fields:

- The new *Value* and *Unit* fields must be *Enumerative* fields of the *Enumeration* data type, and on the *Enumeration values* tab you must mark the *Dynamic list* option, but DO NOT specify a *Dynamic list field*.
- The new *Measure* field must be a linked field to the Thesaurus *Term* field in the fixed domain *Environment*.
On the *Linked field mapping* tab, in the *Copy fields from linked record* list, you must include the linked-to field and the linked field as the source and destination fields if you created a forward reference for this linked field.
In the *Copy lists from linked record* list underneath it, you must include the two *Value* and *Unit* Thesaurus fields as source fields, and their destination counterparts, that you created in the previous steps.

(And finally you must of course adjust or create a screen and include it in your application, to display this data and allow user input.)

Now, when open your application, and access the primary database (in this case the conservation activities database), and edit a new record for example, then on the screen for environmental requirements, you will see something like the following:

The screenshot shows a form titled "Environmental requirements" with three columns: "Measure", "Value", and "Unit". The "Measure" field contains the text "temperature". The "Value" field is a dropdown menu with "17-23" selected and "15-25" highlighted by a mouse cursor. The "Unit" field is also a dropdown menu. Below the "Value" field, there is a "Notes" section with a text input field.

The *Measure* field is validated against the Thesaurus (*Environment* domain), so the user can type a measure or choose one from the *Linked record search screen* with **shift-F4**. The *Value* and *Unit* fields have become drop-down list that are dynamically filled with data from the measure record that the user just chose. So for *temperature* the user will be able to choose from the two specified ranges, and if he or she opens the *Unit* list, the only choice will be *Degrees Celsius*, in this example.

Write fields to linked record

Write-back links were created to allow the user to edit merged-in fields directly in the primary database and have the changes written back to the corresponding fields in the linked database. Source fields are now screen fields in the primary database, while the destination fields are fields in the linked database. Data is written back when you save the primary record. Note that you do not need to define the source fields in the data dictionary of the primary database if the current linked field has a forward reference.

Write-back links should be used sparsely, since the consequences may be serious: after all, the user may change a linked record without him or her knowing it, causing all records in other databases that link to that record with a forward reference, to reflect the changes. And if this linked field has no forward reference, then only the current primary record and the linked record are updated, but not all other primary records.

On the other hand, when you do not specify write-back links, you should make sure that any merged-in fields are not editable.

If you do specify *Write fields to linked record*, then remember to **never** include the currently specified linked field and linked-to field (*Lookup field*) in this list, because that would cause errors due to the fact that writing to the linked database is already part of the functionality for linked fields.

A special case is a linked tag like *ti* in the *Orders* database, which uses the *Lookup field TX* (a term index on the whole *Title* field in *Document*) instead of the *ti* free text index: because the linked tag *ti* must write a title back to the *ti* tag in *Document*, this tag pair should be (and is) included in the write-back fields list.

If one or more of the fields to be written along with a linked field, are linked fields to some other database themselves (let's call them primary linked fields), then you should always write the link reference fields associated with those primary linked fields instead of the primary linked fields themselves; the target fields in the secondary database must then of course also be link reference fields for linked fields. If the primary linked fields are not linked on reference, then you have no choice but to use the linked fields themselves.

If one or more of the fields to be written along with a linked field, are merged-in fields (with some other, primary linked field) in the primary database, then there's no need to write them to the linked secondary database if you already write the relevant (link reference field of the) primary linked field to an identically defined field in the secondary database (including its own merge list).

Further, it is recommended that write-back fields never have a default value in the primary database. If a default value is desired, set this value for the target field in the linked database instead. This does have the minor disadvantage that when you create a new linked record from within the primary database, the default value won't be visible until you store the primary record. This problem does not occur when you link to an existing record.

4.11.7.3 Enumerative field

4.11.7.3.1 Enumeration Values

On the *Enumeration values* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Enumerative field* on the *Field properties* tab, you set up the drop-down list that makes up an enumerative field. In a running application a user must choose a value from this drop-down list for an entry field associated with this data dictionary field; another value cannot be entered.

[Click here](#) for information on how to edit properties in general. On the current tab you'll find the following settings:

Enumeration source

First choose the type of list you want to offer the user: a *Fixed list* or a *List from a record* (a dynamic list). A fixed list is filled only with preset values that you provide on this tab, while a list from a record typically is filled with values from a field in the current record in a running application, that you specify for *List from a record*.

A more advanced use of a dynamic list is also possible: with the *Copy lists from linked record* functionality you are able to display all occurrences of some source field from a secondary database, in a drop-down list of a destination field in a primary database, dependent on the value in an associated linked field. (For this use, the *Field for dynamic lists* must be left empty.)

Sort enumeration values

Normally in a drop-down list, the order of the list of values is

presented as it is entered underneath *Enumeration values* on the current tab. But if such a list is long, it may serve users better if it is sorted alphabetically. To not have to do this sorting manually, you can choose a *Sort enumeration values* type: *Ascending* or *Descending*. You can enter the list of values in a random order; in the running application these values will then be automatically sorted alphabetically (ascending or descending) when the relevant drop-down list is opened.

Field for dynamic lists

If you chose *List from a record* as the *Enumeration source* of the enumerative list and you want to use values from the current record, then here you specify the source tag or field name from which the drop-down list in the running application must retrieve its values when the user opens the entry field associated with this data dictionary field. The source field can only be a local field (from the current record).

This means that the list will be different for each record, and may sometimes only contain one value or even none at all.

Note that of merged-in fields you only have the first occurrence, so you cannot retrieve all occurrences of a linked field this way.

You *can* retrieve all occurrences of a linked field though, via the *Copy lists from linked record* functionality. For this use, the *Field for dynamic lists* must be left empty.) An example of this advanced functionality is when you have one database that holds records for cinemas. In each of those records you list the theatres available in that cinema. These records are linked to records of movies in a movie database. In each movie record you have a linked field (to the cinema database) in which you enter the name of a cinema or retrieve it through a *Linked record search screen*. For the linked field you have defined a *Linked field mapping* to copy all occurrences of the field that holds the theatres in the currently selected cinema to a destination field in your movie record. In the movie database you have specified a field as a dynamic enumerative list that retrieves its values from said destination field through the linked field mapping. Thus, as soon as you select a cinema in a movie record, and click the enumerated field, a drop-down list shows all theatres in that cinema.

Value

If you chose *Fixed list* as the *Enumeration source* of the enumerative list, then in this column you enter the names of the enumerative values. Only these names will be saved in the database, not the language values. But the user will never see these names, so they may be an abbreviation and always in English for instance. You can use upper or lower-case and spaces, if you wish.

Language value

If you chose *Fixed list* as the *Enumeration source* of the enumerative list, then here you provide all possible text values per name value for the field as the user will see them.

4.11.7.4 Auto-numbered field

4.11.7.4.1 Automatic numbering

On the *Automatic numbering* tab, which is present when you have selected a new or existing data dictionary field in a database and set the *Type* of this field to *Automatic numbering field* on the *Field properties* tab, you set up the routine that automatically calculates a number for this field in a new record.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Assignment moment

The *Assignment moment* specifies when the automatic number should be calculated. You can choose from:

- *Never*: this field should not be automatically numbered (default).
- *Before input or edit*: when a record is edited or created, and the auto-numbering field is (still) empty, it will be filled in accordance with the rest of the settings for this field.
- *Before storage*: an automatic number will be assigned only when the record is saved, and the auto-numbering field is still empty.

When the automatic number is generated is not trivial. This is because once generated numbers cannot be assigned to a record again. On the one hand this means that when you remove a record with an automatically generated number from the database, the associated number will never again be assigned automatically to another record. On the other hand this means that when you let Adlib calculate a new number during edit or input, so that you can see the new number as soon as you open a new record, that the number will be "lost" if you decide not to save the new record. If you subsequently open another new record, the associated number is incremented again.

If it is undesirable that an automatically generated number is lost when you do not save a new record, then let numbers be calculated before storage. However, during input or editing no number will then

be visible yet: this will appear only after storage. (But when deleting an already saved record, the associated number will still be lost.)

Automatic assignment

Select this option if users may not write in, or change this field. Every automatic number for this field is unique.

Allow manual input

Select this option if users may enter numbers themselves, or change automatically assigned numbers, and the number may differ from the format string. The field must be given a unique index, as double numbers are not wanted. (If numbers are assigned manually, there is always a chance that the number already exists in another record; this is prevented by the unique index.)

Prefix string

Enter a fixed text (e.g. a general reference code) that is pasted in front of the automatically assigned number.

Start value

Choose the first value (an integer) of the automatically assigned numbers; so auto-numbering does not necessarily have to start at 1. Automatic numbering generates 32-bit numbers, so that these numbers can run as high as record numbers (2,147,483,647).

Increment value

Choose the value with which the automatic number must be incremented for each new record (e.g. 1).

Postfix string

This is a fixed text that is pasted behind the automatically assigned number.

Number format string

A number format string is used to layout the automatically assigned number. See the ADAPL function `str$()` in the ADAPL reference, for the syntax of this string.

Counter

In this option, the current number value for this field, as kept in the

.cnt file for the current database, is displayed. You can change this value, and then click the *Apply* button to set the changed value in said file. This is handy when you've deleted one or more most recent records and want the automatic numbering to continue from the lower highest number, for instance.

The .cnt file

For every existing CBF database, one extra file (with the name of the database and the extension .cnt) is created in your *\data* directory, in which the most recent (and thus highest) automatically calculated number, and any automatically assigned numbered file name for an Image field, are stored per field. When a number is about to be assigned, this .cnt file will be locked, a new number is calculated and added by replacing the old number in there, and the lock is removed. This guarantees that all new numbers are unique, and prevents a number from being assigned twice if two records are modified simultaneously. Normally, these files shouldn't be edited manually. For Adlib SQL Server and Adlib Oracle databases, automatic numbering information is stored in tables (in the database), not as separate files.

4.11.7.5 Image field

4.11.7.5.1 Image field properties

On the *Image field properties* tab, which is present when you have selected a new or existing data dictionary field in a database and on the *Field properties* tab set the *Type* of this field to *Normal* and the *Data type* to *Image*, you can set which method you want to use for saving images that you've loaded through Windows Image Acquisition and you may set up a routine that automatically calculates a number and generates a file name for the new image. You can also use this tab to set which part of paths to linked images, movie files, mp3s or other media files, in general must be hidden from the user and to which location images or other media files must be copied when you link them.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

File name assignment type

Choose *From field content* if you want the user to be able to manually

enter a name under which the image to be read through WIA, must be saved. You cannot set any of the options under *File name generation* then.

Choose *Automatic sequential generation* if you want to automatically generate file names for images that are being read in by the user, by means of an auto-numbering format that you specify in the options below.

Prefix string

Enter a path (absolute, or relative to the application folder) under which images created through WIA must be saved. You can use a relative path to hide the location of the image directory from users. However, do not enter a path here if you intend to use the *Storage path* option below. Instead of a path, you may also provide a partial string with which the file name should start.

This prefix string is the first part of the complete string that will be built up to automatically save an image file and refer to it. Examples:

```
..\images\img  
../images/  
C:\Adlib files\photos\
```

Start value

Choose the first value (an integer) of the automatically assigned numbers that will be the middle part of the image file names; so auto-numbering does not necessarily have to start at 1.

Automatic numbering generates 32-bit numbers, so that these numbers can run as high as record numbers (2,147,483,647).

Increment value

Choose the value with which the automatic number that will be the middle part of the image file name, must be incremented for each new image that is read (e.g. 1).

Postfix string

This is a fixed text that is pasted behind the automatically assigned number, and it forms the closing part of the complete string that will be built up to automatically save an image file and refer to it. So you must enter (at least) the extension of the file type to be generated, preceded by a dot. Examples:

```
-blackandwhite.gif
```

-2005.jpg

.bmp

Which images formats can be generated, depends on your WIA device.

Number format string

A number format string is used to lay out the automatically calculated number that forms the heart of the automatically assigned file name. See the ADAPL function `str$()` in the ADAPL reference, for the syntax of this format string.

Storage type

This option allows you to explicitly indicate the storage method for an image or other media file to be linked in Adlib. This is optional for *File system* and *URL*, since Adlib itself is already capable to make the distinction between fixed paths and URLs, but it is mandatory for a call to Adlib Ingest*.

* Adlib Ingest is a tool that, to this end, can be used to add currently linked images directly to a DAMS (Digital Asset Management System). Adlib Ingest is currently (January 2014) still under development.

To ensure backwards compatibility, the option defaults to *Undefined*. Urls and fixed paths in the *Storage path* option will automatically be recognized by Adlib. So you don't have to make any changes to your current system. However, you may improve system performance during the linking of images slightly by setting a storage type for existing image fields now, corresponding to the type of the already set *Storage path* of course. If *Storage path* is empty, *Storage type* should have been set to *Undefined*.

Storage path

This option provides a way to automatically store all linked images or other media files (both existing files, and newly created images through WIA) in a predetermined location, and hide the path to the files from users. This means that when, in this field in the running application the user selects e.g. an existing image file anywhere on the computer or network (through the *Find image file* button or by typing the path manually) it will be copied (!) to the folder set in the *Storage path* option (unless the file already exists in this folder, in which case the user has to confirm his or her choice). Only the file name (without preceding path) must remain visible in the field. A newly created linked image will be stored in the storage path folder directly, without leaving an original file elsewhere.

The copying or direct storage of all linked media files to a single folder ensures that the path for all links is controlled by the *Storage path*; preferably choose a UNC path as the fixed path (if you are not addressing a web service with a URL), since local paths are only relevant to the individual computer from which they originate.

Because of the way this functionality is constructed, you cannot use this option to store media files in different subfolders of the main target folder: the option effectively allows users to select media files anywhere on the system or network, after which they will be copied to the single target folder. Only the file name remains visible in the field. On retrieval of the media file, either the *Retrieval path* (which takes precedence, if filled in) or the *Storage path* will be used to put together the entire path to the image. If neither *Retrieval path* nor *Storage path* have been filled in, the record itself should contain the full path to the image file.

The *Storage path* is basically constructed as follows:

```
<path or URL><%data%>
```

Everything in between < and > is optional, and what is written between percent characters must be entered literally. For the path or URL you enter either the path to the folder in which the media files linked in the current field must be stored or a URL to a web service which can store media files (using e.g. `wwwopac.ashx`); this path or URL will be hidden from the user, which improves the security of your data.

The `%data%` part in the format string is necessary for Adlib to include the file name from the current field in the string. Examples of *Storage paths* are:

```
\\our_server3\documents\%data%
```

```
http://server/documents/%data%
```

```
http://www.ourmuseum.com/Adlib/wwwopac.ashx?
```

```
command=writecontent&server=adlibimages&value=%data%
```

If you use `wwwopac.ashx` to store and retrieve images, the *Storage path* and *Retrieval path* will have to contain different URL's because of the included save or retrieve action, and other processing instructions. Note that if you use `wwwopac.ashx` to retrieve images, it is recommended to also use it to store images: storing images to a static path while retrieving them via `wwwopac.ashx` may lead to an error 123 when linking a new image.

Contrary to writing an image file using `wwwopac.ashx` outside of an `adlwin.exe` application, you do not have to provide a `bytes` parameter here: `adlwin.exe` automatically creates a binary file from the image file it finds in the `%data%` variable and sends it along with the `wwwopac.ashx` request to write the image.

See the Adlib API website for more information about image retrieval with `wwwopac.ashx` and setting up an image server.

Retrieval type

This option allows you to explicitly indicate the retrieval method for an image or other media file to be linked in Adlib. This is optional for *File system* and *URL*, since Adlib itself is already capable to make the distinction between fixed paths and URLs.

To ensure backwards compatibility, the option defaults to *Undefined*. Urls and fixed paths in the *Retrieval path* and *Thumbnail retrieval path* options will automatically be recognized by Adlib. So you don't have to make any changes to your current system. However, you may improve system performance during the retrieval of images slightly by setting a retrieval type for existing image fields now, corresponding to the type of the already set (*Thumbnail*) *Retrieval path* of course. If both paths are empty, *Retrieval type* should have been set to *Undefined*.

Retrieval path

The *Retrieval path* - available from Adlib Designer 6.5 - provides a way for Adlib for Windows applications and Internet Server web applications to retrieve a linked media file and display (or play) it in Adlib's Media Viewer component (which may be available in the detailed presentation of an Adlib record) or Windows Media Player on a web page. Of the linked media file, only the file name (usually without path information) has been stored in the linked field. For this purpose, either a local (network) path, a call (URL) to `wwwopac.ashx` (or to the older `.exe` version, only for images though) can be entered here, to allow Adlib to retrieve images or other media files for the detailed record presentation. (Note that in fields of the *Application* data type, in which other file types such as documents have been linked, of the three *path* options here, currently only this *Retrieval path* option can be used, to hide the path to the file.)

This means that on retrieval (this option not for storage), the file name will be merged with the *Retrieval path* to form a complete path. The *Retrieval path*, which may include possible sensitive information needed to access the media file, will be hidden from the user. Any processing of an image* on retrieval, like scaling, as can be done with `wwwopac`, is executed server-side: this may improve performance.

The syntax of retrieval paths to static (local or network) folders is the same as that of *Storage path*, namely:

```
<path><%data%>
```

Examples of static and dynamic paths:

```
C:\Our images\%data%
```

```
\\ourserver\d-drive\ourimages\%data%.jpg
```

```
http://www.ourmuseum.com/Adlib/wwwopac.ashx?
```

```
command=getcontent&server=adlibimages&value=%data%
```

In the case of a static path, the *Retrieval path* itself must be identical to the *Storage path* (if *Storage path* has been specified, which is not mandatory): after all, media files will have to be retrieved from the same location as where they are stored. However, if you leave the *Retrieval path* empty while the *Storage path* option is filled with a static path, then Adlib will implicitly use the *Storage path* also to retrieve media files. If neither *Retrieval path* nor *Storage path* has been filled in, the record itself should contain the full path to the media file.

On the other hand, if you use `wwwopac.ashx` to store and retrieve images or other media files, the *Storage path* and *Retrieval path* will have to contain different URL's because of the included save or retrieve action, and other processing instructions. Note that if you use `wwwopac.ashx` to retrieve media files, it is recommended to also use it to store them: storing media files to a static path while retrieving them via `wwwopac.ashx` may lead to an error 123 when linking a new file. (See the Adlib API website for more information about image retrieval with `wwwopac.ashx` and setting up an image server.)

If you haven't upgraded to `wwwopac.ashx` yet, you may also use a `wwwopac.exe` URL to retrieve image files (no other media files). Storing of images by `wwwopac.exe` is not possible though, so in *Storage path* a local or network path could be used instead, for example. A `wwwopac.exe` URL (with the .exe-specific `outputtype` parameter) to retrieve an image, could look as follows:

```
http://publicserver/wwwopac.exe?thumbnail=\\ourserver\images\%data%&outputtype=image/jpeg
```

On processing of the `wwwopac.exe` URL, `%data%` will be substituted by the contents of the image field (a file name or partial path). See the WWWOPAC Reference guide for details about the syntax of such `wwwopac.exe` URLs.

* Back to retrieving media files using the current `wwwopac.ashx`: you may notice that it allows you to specify image file specific arguments in the `getcontent` call (for example to scale all retrieved images) which cannot be applied to movies or sound files. Of course, if you only ever retrieve images there's no problem, but if you retrieve images as well as other media files and you have applied `getcontent` arguments other than `server` and `value`, then `wwwopac.ashx` will attempt to load an image file with the same id but with the `.jpg`

extension, instead of the requested movie or sound file. If that image file is not present, the API tries to load a default image named *movie.jpg*. If that fails as well, a system exception will be thrown. So if you want to use a single query to retrieve both images and movies while applying any of the image specific `getcontent` arguments (as may be the case for the current *Retrieval path* setting for an image field), do at least make sure a default *movie.jpg* image is present to avoid the exception.

If you use scaling options here in the *Retrieval path* to reduce the size of retrieved images, it may improve performance of Adlib whenever a large image is displayed but take into account that the reduced image contains less information than the original and is no longer identical to that original. This means that the user can't zoom in as much as the original would allow and that saving the displayed image to a different location from within the Adlib Media Viewer would save a copy of the scaled image file, not a copy of the original. If you do want the original image to be displayed in Adlib, then retrieve it by just using the `server` and `value` arguments of the `getcontent` command. Saving a copy of the original image from within the Adlib Media Viewer component would mean creating an exact identical copy (including any metadata), as long as you maintain the original image file type: if you would save the copy using a different file extension, the copy would no longer be identical to the original.

In web applications it may also be desirable to send the current user name and password along with the request to open the clicked file, so that certain users can be excluded from opening the file. Use `%username%` and possibly `%password%` behind `%data%` to enable this. If you use standard Windows authentication, only the current user name (Windows login name) can be sent along. If the string also contains `%password%`, then that part will remain empty. If you have activated the *Adlib pbk* -, *Adlib database* -, *Active directory* -, or *HTTP* user authentication, then both the user name and password can be sent along.

Example:

```
http://ourserver/wwwopac.exe?thumbnail=%data%%username%  
password%&outputtype=image/jpeg
```

Although we recommend to store all images in a single folder - there is usually no need to structure your images folder into subfolders because with Adlib you search images by means of their metadata, not their physical location - you may still want your images to be grouped in subfolders anyway, because someone needs to be able to find and edit them outside of Adlib. In that case you can't use the *Storage path* option, to begin with, so you'll have to organize your

images in a folder structure manually, using Windows Explorer for example. Further, if you still want to use a *Retrieval path*, then when you link images from these subfolders to reproduction records, you'll have to edit each automatically entered path to the image so that it only contains the proper subfolder path to the image and the image file name. The combination of the *Retrieval path* set here and the <subfolder>\<file_name> data in records should then make up the entire path.

Thumbnail retrieval path

This option basically offers the same functionality as the option above. The *Thumbnail retrieval path* provides a way for Adlib for Windows applications and Internet Server web applications to retrieve thumbnail images of linked media files and display them in Adlib's Media Viewer component and on Brief display screens. Of each linked media file, only the file name (usually without path information) has been stored in the linked field. (If no *Thumbnail retrieval path* has been specified, Adlib will check the *Retrieval path* option above and if that hasn't been specified either, the *Storage path*, to retrieve an image from the proper location: if none of these options have been filled in, the record itself should contain the full path to the image file.) You are offered the (*Thumbnail*) *retrieval path* option twice under a different name to allow you to set different retrieval paths for 'normal' sized images and for thumbnail images, since Adlib applications typically use both. Especially when your 'normal' sized images are actually quite large files and you are not yet using *wwwopac* to retrieve pre-scaled thumbnails (see the next paragraph), you may find there is a performance toll to be paid when *adlwin.exe* has to retrieve and scale those large images to thumbnails. If you're still not looking to use *wwwopac*, the solution would be to manually create small(er) versions (not necessarily exactly the target size) of all your large image files in advance (using photo editing software for example), put those scaled images in a separate images folder and refer to that folder in the *Thumbnail retrieval path*. This could increase performance substantially.

However, the *wwwopac.ashx* and *wwwopac.exe* both offer functionality to scale images automatically, so you don't have to create thumbnails manually first. An added advantage to scaling images to be retrieved using a (thumbnail) retrieval path is that they are scaled on the server, instead of on the client. This means that smaller images (smaller files) need to be sent over the internet or local network, which improves the performance of your application. Two example *wwwopac* URLs for scaling the retrieved image (the first specific to *wwwopac.exe*, the second specific to *wwwopac.ashx*, each with their own parameters), are the following:

```
http://publicserver/wwwopac.exe?thumbnail=\\ourserver\images\%
```

```
data%&outputtype=image/jpeg&xsize=3030
http://www.ourmuseum.com/Adlib/wwwopac.ashx?
command=getcontent&server=adlibimages&value=%data%
&width=100&height=200
```

You may notice that although `wwwopac.ashx` allows you to retrieve different types of media files, it still allows you to specify image file specific arguments in the `getcontent` call (for example to scale all retrieved images to obtain thumbnails) which cannot be applied to movies or sound files. Of course, if you only ever retrieve image thumbnails there's no problem, but if you retrieve images as well as other media files and you have applied `getcontent` arguments other than `server` and `value`, then `wwwopac.ashx` will attempt to load an image file with the same id but with the `.jpg` extension, instead of the requested movie or sound file and create a thumbnail of the image file. If that image file is not present, the API tries to load a default image named `movie.jpg`. If that fails as well, a system exception will be thrown. So if you want to use a single query to retrieve both images and movies while applying any of the image specific `getcontent` arguments (as may be the case for the current *Thumbnail retrieval path* setting for an image field), do at least make sure a default `movie.jpg` image is present to avoid the exception.

Note that a further advantage of `wwwopac.ashx` over the older `wwwopac.exe` is that `wwwopac.ashx` automatically creates cache folders in which scaled images will be stored: this provides another performance boost because now an image only needs to be scaled once to a particular size. After that, the next time the same image with the same scaling is requested, it will automatically be retrieved from the relevant cache folder directly.

Also note that `adlwin.exe` (the software that runs your local Adlib application) will scale the image retrieved through `wwwopac` again (if necessary), to the dimensions of the thumbnail display as set on the relevant brief display screen. So if thumbnails on the brief display would have been set to 50x50, and you would first scale an image to 100x100 on the server via a URL in the *Thumbnail retrieval path*, then `adlwin.exe` would scale it further to 50x50. So, retrieve images for thumbnail display preferably in the dimensions in which they should be displayed on list screens: it saves processing time. To always automatically retrieve thumbnails in the dimensions in which they have to be displayed on the list screen, you must replace the fixed dimension values in the URL by the `%width%` and `%height%` variables, for example:

```
http://www.ourmuseum.com/Adlib/wwwopac.ashx?
command=getcontent&server=adlibimages&value=%data%&width=%width%
&height=%height%
```

If you also scale the image to be retrieved for detailed display, via the URL in the other *Retrieval path* option, you may consider using the `%width%` and `%height%` variables in that URL as well. This is because `adlwin.exe` also scales images for detailed display in the Media Viewer, to the dimensions of the Media Viewer window to be precise. However, by using `%width%` and `%height%`, the image will already be retrieved in those dimensions.

See the Adlib API website for information about setting up an image server for `wwwopac.ashx`.

Inpoint tag

Inpoint and outpoint tags are used by the `adlwin.exe` functionality that allows the user to mark start and end points of a fragment in a movie or audio file linked to a record, after which just the fragment can be played (instead of the entire media file). This way the user may create records each describing a specific fragment in the linked media file.

To enable this functionality and activate the relevant *Set start point* and *Set end point* buttons in the *Media* context menu in the Adlib ribbon when the Media Viewer is present, you must create two new, non-repeatable numerical fields in the database in which you'd like to described media fragments. The tag of the start point field must be entered in the current image field property. The field must also be added (non-repeatable, writable and numerical) to a detail screen for the relevant database, preferably one holding the currently edited image field.

Outpoint tag

Inpoint and outpoint tags are used by the `adlwin.exe` functionality that allows the user to mark start and end points of a fragment in a movie or audio file linked to a record, after which just the fragment can be played (instead of the entire media file). This way the user may create records each describing a specific fragment in the linked media file.

To enable this functionality and activate the relevant *Set start point* and *Set end point* buttons in the *Media* context menu in the Adlib ribbon when the Media Viewer is present, you must create two new, non-repeatable numerical fields in the database in which you'd like to described media fragments. The tag of the end point field must be entered in the current image field property. The field must also be added (non-repeatable, writable and numerical) to a detail screen for the relevant database, preferably one holding the currently edited image field.

The .cnt file

For every existing database, one extra file (with the name of the database and the extension *.cnt*) is created in your *\data* directory, in which the most recent (and thus highest) automatically assigned numbered file name and other automatically calculated numbers are stored per field. When a file name is about to be assigned, this *.cnt* file will be locked, a new file name is generated and added by replacing the old file name in there, and the lock is removed. This guarantees that all new numbered file names are unique, and prevents a file name from being assigned twice if two records are modified simultaneously.

4.11.7.6 Application field

4.11.7.6.1 Application field properties

On the *Application field properties* tab, which is present when you have selected a new or existing data dictionary field in a database and on the *Field properties* tab set the *Type* of this field to *Normal* and the *Data type* to *Application*, you set save options for documents to be linked and you may also set up a routine that automatically calculates a number and generates a file name for the copied document; you can also set which part of paths to linked files in general must be hidden from the user.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

File name assignment type

Choose *From field content* if you want the user to be able to manually enter a name under which a copy of the document to be linked, must be saved. You cannot set any of the options under *File name generation* then.

Choose *Automatic sequential generation* if you want to automatically generate file names for documents that are being linked by the user, by means of an auto-numbering format that you specify in the options below.

Prefix string

Enter a path (absolute, or relative to the application folder) under which copies of linked documents must be saved. You can use a

relative path to hide the location of the document directory from users. However, do not enter a path here if you intend to use the *Storage path* option below. Instead of a path, you may also provide a partial string with which the file name should start. This prefix string is the first part of the complete string that will be built up to automatically save a file and refer to it. Examples for *File system*:

```
..\documents\pdf  
../files/  
C:\Adlib files\linkeddoks\
```

Start value

Choose the first value (an integer) of the automatically assigned numbers that will be the middle part of the file names; so auto-numbering does not necessarily have to start at 1. Automatic numbering generates 32-bit numbers, so that these numbers can run as high as record numbers (2,147,483,647).

Increment value

Choose the value with which the automatic number, that will be the middle part of the file name, must be incremented for each new image that is read in (e.g. 1).

Postfix string

This is a fixed text that is pasted behind the automatically assigned number, and it forms the closing part of the complete string that will be built up to automatically save a file and refer to it. So you must enter (at least) the extension of the file type to be generated, preceded by a dot. Examples:

```
-copy.doc  
-2008.pdf  
.docx
```

Number format string

A number format string is used to lay out the automatically calculated number that forms the heart of the automatically assigned file name. See the ADAPL function `str$()` in the ADAPL reference, for the syntax of this format string.

Storage type

This option allows you to explicitly indicate the storage method for a file to be linked in Adlib. This is optional for *File system* and *URL*, since Adlib itself is already capable to make the distinction between fixed paths and URLs, but it is mandatory for a call to Adlib Ingest*.

* Adlib Ingest is a tool that, to this end, can be used to add currently linked images directly to a DAMS (Digital Asset Management System). Adlib Ingest is currently (April 2013) still under development.

To ensure backwards compatibility, the option defaults to *Undefined*. Urls and fixed paths in the *Storage path* option will automatically be recognized by Adlib. So you don't have to make any changes to your current system. However, you may improve system performance during the linking of files slightly by setting a storage type for existing application fields now, corresponding to the type of the already set *Storage path* of course. If *Storage path* is empty, *Storage type* should have been set to *Undefined*.

Storage path

This option provides a way to automatically store all linked files in a predetermined location, and hide the path to the files from users. This means that when, in this field in the running application the user selects an existing file anywhere on the computer or network (through the *Find file* button or by typing the path manually) it will be copied (!) to the folder set in the *Storage path* option (unless the file already exists in this folder, in which case the user has to confirm his or her choice). Only the file name (without preceding path) must remain visible in the field.

The copying of all linked files to one folder ensures that the path for all links is controlled by the *Storage path*; preferably choose a UNC path as the fixed path, since local paths are only relevant to the individual computer from which they originate.

The *Storage path* is basically constructed as follows:

```
<fixed text (path)><%data%>
```

Everything in between < and > is optional, and what is written between percent characters must be entered literally. For the fixed text you enter the path to the folder in which the files linked in the current field must be stored; this path will be hidden from the user, which improves the security of your data.

The %data% part in the format string is necessary for Adlib to include the file name from the current field in the string. Examples of *Storage paths* are:

```
\\our_server3\documents\%data%
```

```
G:\server\documents\%data%
```

Retrieval path

The *Retrieval path* - available from Adlib Designer 6.5 - provides a way for Adlib to retrieve a linked file of which only the file name without path information has been stored in the linked field. For this purpose, a local (network) path can be entered to retrieve files.

This means that on retrieval, the file name will be merged with the *Retrieval path* to form a complete path. The *Retrieval path*, which may include possible sensitive information needed to access the image, will be hidden from the user.

The syntax of retrieval paths to static (local or network) folders is the same as that of *Storage path*, namely:

```
<fixed text (path)><%data%>
```

Examples:

```
C:\Our files\%data%
```

```
\\ourserver\d-drive\ourfiles\%data%.pdf
```

The *Retrieval path* must be identical to the *Storage path*: after all, files will have to be retrieved from the same location as where they are stored.

Thumbnail retrieval path

<This option does not apply to application fields.>

The .cnt file

For every existing database, one extra file (with the name of the database and the extension *.cnt*) is created in your *\data* directory, in which the most recent (and thus highest) automatically assigned numbered file name and other automatically calculated numbers are stored per field. When a file name is about to be assigned, this *.cnt* file will be locked, a new file name is generated and added by replacing the old file name in there, and the lock is removed. This guarantees that all new numbered file names are unique, and prevents a file name from being assigned twice if two records are modified simultaneously.

4.11.7.7 Multi language fields

On the *Multi language fields* tab, which is present when you have selected a new or existing data dictionary field in a database, you may specify language tags for the standard field **name** of the current field definition.

(Note that there are two ways of implementing multi-lingual fields.) Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Language tags

Only if want your application to be multilingual, enter here a unique tag for every language other than English, in which you wish to be able to enter data. Each language tag must be a unique tag in this database. (Because English is the standard language, the *English* language tag is the same as the *Field tag* specified on the *Field properties* tab.)

4.11.7.8 User interface texts

On the *User interface texts* tab, which is present when you have selected a new or existing data dictionary field in a database, you may specify label texts for screen fields and/or menu texts for methods that link to this field. In most existing Adlib applications, these texts are specified in the *Screen editor* on entry field level, in the *Label* properties, and respectively in the application setup on method level, in the *Menu texts*. To prevent redundancy in new applications or when adding fields to existing applications, you could enter the user interface texts on data dictionary field level, to specify these texts only once.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Screen texts

Enter in as much languages as you want to make your application available in, the texts that all labels should have that belong to screen entry fields that are associated with the current data dictionary field.

The screen texts you define here won't be copied to the *Label* properties though. But the screen texts you define for a data dictionary field will appear in the running application if the *Label* properties for the concerning entry field are **empty**. If the label texts in the *Label* properties have been filled in, those texts will take precedence.

Method texts

Enter in as much languages as you want to make your application available in, the texts that methods that search this data dictionary field should have in the access points list (step 2) of the *Search wizard* in a running application.

When you create a new method in the application setup, and you specify a *Search field*, the *Method texts* that you specify for the current option will be copied to the *Menu texts* of the new method. There you can change those *Menu texts* if desired: they will take precedence over the *Method texts* you specify for the data dictionary field. (The *Method texts* will not be changed if you edit the *Menu texts*.)

4.11.7.9 Default values

On the *Default values* tab, which is present when you have selected a new or existing data dictionary field in a database, you determine what text (if any) should be placed by default in an entry field associated with this data dictionary field, in a new record or in a new occurrence of this field in any record in a running application. The current interface language (not the data language), before opening the new record or inserting the new occurrence will determine which translation of the default value will be used.

Note that the user can change any default value when editing the record.

Further, default values are only filled in if a field or a new field occurrence is empty - they will not overwrite an existing value.

And you should only set a default value when you know that users will find it handy or helpful.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Type

Choose the type of default value you want for this field. If you choose the *Current date* (date of record input: the system date), the *Current time* (time of input: the system time), or the *User name* (the Windows login name of the user filling in a record with this field in the running application), then you cannot fill in any translations underneath *Text*. If you want to provide a specific text, you should set the *Type* to *Value*, and then specify (translations of) the default texts underneath *Text* in each language you want the application interface to be available in. The functionality is independent of whether the field is multilingual or not, it is only dependent on the interface language. Choose the *Type None*, if you do not want to define a default value for this field.

Text

If you have set the *Type* of the default value to *Value*, then specify the default texts in each language you want the application to be available in. This works as follows:

- If you only specify an English *Text*, this will be the default value for all interface languages.
- If you specify an English *Text* and, let's say, a Dutch *Text*, then when the interface language is e.g. French or German when you open a new record, the English default *Text* will be entered. If the interface language is Dutch, the Dutch *Text* will be entered and for English the English translation will be entered.
- If you only specify a single non-English *Text*, e.g. Dutch, then this translation will only be entered when that interface language (e.g. Dutch) has been selected. In all other languages, no default value will be entered.
- A field may be enumerative and have default values as well; then the default values must be valid neutral values for the enumerative field. An enumerative field definition already contains translations, but the so-called neutral value is the value that is actually stored in the record: users do not get to see this value, only its translations. So if you want to set an item from an enumerative list as the default value for all interface languages, you must specify its neutral value as the English *Text* only.

4.11.7.10 Z39.50 values

On the *Z39.50 values* tab, which is present when you have selected a new or existing data dictionary field in a database, you set up the use of this field in the Adlib Z39.50 server. For information on Z39.50 see: <http://www.loc.gov/z3950/agency/>

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Use attribute

<no Help available yet.>

GRS-1 tag path

<no Help available yet.>

Marc tag

<no Help available yet.>

4.11.7.11 Access rights

On the *Access rights* tab, which is present when you have selected a new or existing field in a database, you determine the access rights for this field, to restrict access to records, dependent on the user (login name in Windows) and its assigned role.

Click here for information on how to edit properties in general. On the current tab you'll find the following settings:

Access

Here you may define which *Roles* have which *Access rights* to this field. You can indicate for each role whether no access (*None*), *Read* access, *Write* access or *Full* access must apply. If a role is not linked to this field, then each user linked to that role has full access by default. A user without a role always has full access. Users are assigned to roles in the application setup.

See also

Security in Adlib

4.11.8 Internal links

4.11.8.1 Internal link properties

On the *Internal link properties* tab, which is present when you have selected a new or existing internal link in a database, you set up this internal link between two or more existing data dictionary fields.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Field

Enter or select the term tag or field name of the normal field on which you want to base this internal link. Base each internal link in a database on the field in which a keyword is defined in each record in that database. Typically, in the *thesau* file this is *te* (*term*), and *BA* (*name*) in *people*.

Type

Choose the type of link you want to make. There are five possible types. For any type you have to fill in the appropriate tags in the accompanying *Relation tags* on this tab (each type of course has different associated *Relation tags*):

1. **Hierarchical (type 1) - broader and narrower:** for this type, in the *Broader field* and *Narrower field* options, enter existing tags or field names from the current database, in which broader and respectively narrower terms are saved.
2. **Related (type 2):** for this type, in the *Related field* option, enter an existing tag or field name from the current database, in which related terms are entered.
3. **Equivalent (type 3):** for this type, in the *Equivalent field* option, enter an existing tag or field name from the current database, in which equivalent terms are entered.
4. **Preferred (type 4), and non-preferred:** for this type, in the *Use field* and *Used-for field* options, enter existing tags or field names from the current database, in which preferred and respectively non-preferred terms are saved.
5. **Semantic factoring (type 5):** for this type, in the *Semantic factor field* and *Semantic factor-for field* options, enter existing tags or field names from the current database, in which semantic factors and respectively concept terms are saved.

Sort values

With this option, Adlib sorts the terms entered in different occurrences of the linked fields that make up this internal link definition, either *Ascending* (0-9, a-z), or *Descending* (z-a, 9-0) before saving a record in this database. The linked fields mentioned here, are the *Broader term field* and the *Narrower term field* set in the options below. Choose *Unsorted* if you want to leave terms in said fields in the order in which you entered them or if the fields that make up this internal link definition are multilingual fields.

In the properties of any field you may also set an *Occurrence sort order*. This may result in conflicting sort instructions if you set both sort options differently for internally linked fields. But Adlib simply first executes any sorting set in the internal link definition and after that, any sorting set in the field properties.

Database scope

Choose whether the automatic updating of internally linked fields after saving a record with such links must be done throughout the entire *Database* which holds the specified tags on this tab, or only in

the *Dataset* in which the user currently saves a record. More specifically, the **current** dataset is the dataset (called data source in the general user guide) opened by the user explicitly in *Step 1* of the *Search wizard* in Adlib before editing a record. This is relevant when a record is part of more than one dataset, which for example would be the case for a book which is part of both a super-dataset called *fullcatalogue* (encompassing the whole database) and a sub-dataset called *book*; a hierarchical dataset structure is also common in so-called enterprise edition Adlib applications.

If you allow the user to create new linked records from within an internally linked field, and you want the internally linked record to be saved in the same dataset as the current record, then make sure you set this option to *Dataset*. To make sure that internally linked records will be stored in the current dataset you should also set the *Folder* property of all linked fields which are part of the internal link setup in the current database to a single dot (instead of *../data*) and the *Database* property below it should literally be set to *=*. The *Dataset* property of those linked fields should not be set: setting that property would mean that the linked record will always be stored in the set dataset instead of the actual current one.

In our standard applications (version 4.2 and older), the two databases that are usually being associated with having internal links, *thesau* and *people*, have no datasets by default, so the option is irrelevant for these files. In standard applications 4.4 and newer, all databases have at least one dataset, the purpose of which is to have a link structure in place that is prepared for any upgrade to an enterprise edition application, should it ever come up: without going into too much detail, when links already include a dataset, instead of just the database name, it's easier to ensure a proper data migration to an enterprise model in which even the thesaurus can have multiple datasets.

The *collect* database however, does always have datasets and internal links. In our standard applications, the database scope for these internally linked fields is still set to *Database* though, because the user is not allowed to create new linked records anyway; and when internally linking to existing records, it *is* allowed to link to objects from the other dataset(s).

A practical example of this option is the following: suppose you have a library catalogue that is divided into books and serials datasets (amongst others), and you want to continue registering a magazine under a different name than before. You decide to implement this through a broader/narrower internal link in the catalogue database, where the new name will become the broader name of the old name. Because you want the changes you make to the name of the magazine only to have effect on records in the serials dataset, you set the *Database scope* for this internal link to *Dataset*.

Another example would be an enterprise edition of an Adlib application. In such an application, different branches of an organisation all use the same centralized databases, but most or all of those databases are separated into branche-specific datasets, so that each branch always only handles data within its own dataset in the central database. In this case, authority files like the *Thesaurus* and *Persons and institutions* are also separated into datasets and you might want to set the *Database scope* property to *Dataset* to make sure that internally linked records are all saved in the same dataset. (On a side note, this would mean that a term could occur more than once in the database, so then you cannot have a unique index on the relevant field.) On the other hand, if you want a record from a sub-dataset to be able to link internally to other records within a larger super-dataset that encompasses the entire database, even though the new term is created within its own sub-dataset, then set the *Database scope* option to *Database* or leave it at *Undefined*. If you have a database which has only one super-dataset, you can choose either *Database scope*, but to set it to *Dataset* is the safer option in preparation for a possible future addition of datasets, should those be required.

Format string

From 6.0, in the detailed presentation of records from hierarchically built databases like an archive database, the *Thesaurus*, or *Persons and institutions*, you can display the tree structure of all records that are linked internally to the currently displayed record through broader and narrower terms. (See the Adlib User guide for the user aspects of this functionality.)

By default, the tree view displays all narrower and broader terms of the current term. However, for some hierarchical databases you want to be able to set which fields must be displayed in the tree structure. Because a document number in an archive for instance, is probably not very informative in such a structure.

In the current option, for an internal link definition of type 1 (broader and narrower) only, you can make said setting. This format string is constructed as follows:

```
%<tag><[occ]>%<fixed text>...
```

The part between percent characters is the data part: provide the tag that you want to display in the hierarchy. If you do not enter an occurrence number, by default the first occurrence will be used. Behind the data part you may (optionally) provide a fixed text, e.g. explanatory, or to introduce the next data part, since you may repeat `%<tag><[occ]>%`, with or without `<fixed text>` in between. Examples of filled in format strings are:

%te%; non-preferred term: %uf%

%te[2]%

Note that you cannot use field names. And both the data and text parts are optional. If you do not provide a *Format string*, by default the field will be displayed on which the internal link has been defined.

See also

Internal links

The Adlib User Guide (for information about hierarchical relations in the *Thesaurus* and *Persons and institutions*)

4.11.9 Feedback databases (references)

4.11.9.1 Feedback database properties

On the *Feedback database (properties)* tab, which is present when you have selected a new or existing feedback database reference in an authority database, you specify said reference by providing name and path to a primary database.

Click here for information on how to edit properties in general. And click here to read about how to manage objects in the tree view in the *Application browser*. On the current tab you'll find the following settings:

Folder

This property contains the full or relative path to the database you want to set up here as feedback database, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the path to that folder is automatically entered here.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you want to link to. Do not enter the extension of the file. Examples of database names are `DOCUMENT`, `COPIES`, and `Collect`.

Important: when you work in a copy of your live application, then make sure you search the right folder (in the copy) for the proper file: otherwise the relative path will be incorrect when you place back this copy as your live application later on.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some

databases have no datasets.) Selecting a dataset is optional, you can also just link to an entire database.

5 Using ADAPL

§

5.1 Programming in ADAPL

§

5.1.1 Introduction

ADAPL is a general-purpose programming language that incorporates an Adlib record interface. An ADAPL program (also called an "adapl") can either be executed as a stand-alone program or be used during various database functions in Adlib, e.g. to validate screen input, or to carry out an extra check during deletion.

If an ADAPL program is run from within an Adlib application, all data in the current record will be available for manipulation.

ADAPL programs can be used to modify or validate Adlib data or to generate data output routines. Adapls can also be used to start external programs or routines from within an Adlib application.

An ADAPL program is created as an ASCII text file with the extension *.ada*. You can do this in any text editor. After writing the code you must compile the program with the ADAPL compiler to make the code ready for execution.

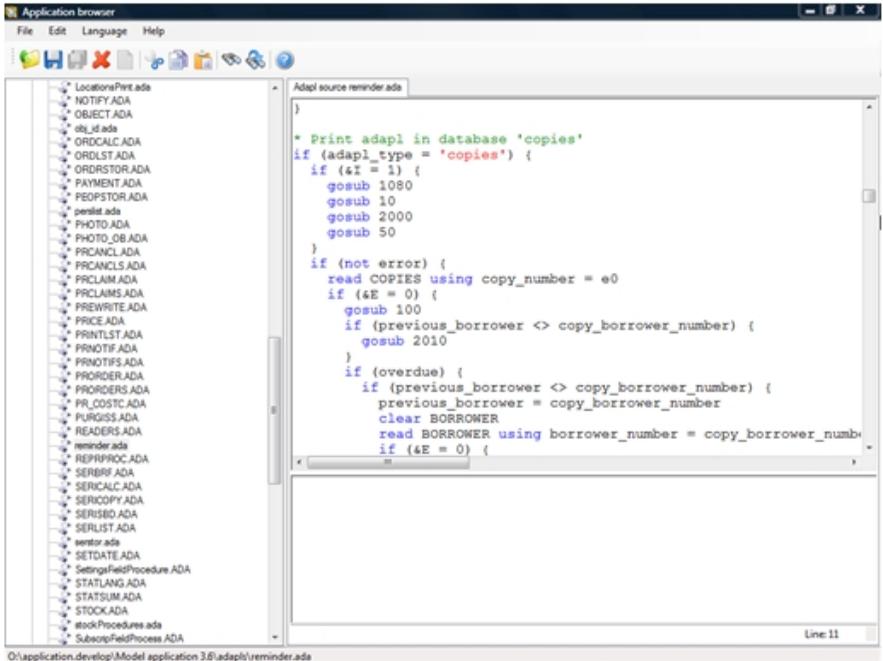
From 6.5.0, you can edit existing adapls in Designer as well: just click an adapl in the tree view in the *Application browser* to open it, and right-click the ADAPL code to open a pop-up menu with the options to save your changes or to compile the file (to the same folder).

Note that when an *.ada* source file is saved/converted to Unicode in UTF8 encoding, a UTF header of three characters is added to the beginning of that file. That first line of adapls often contains an asterisk, to indicate comments. So after conversion, the first line contains the UTF8 header followed by an asterisk.

5.1.2 The ADAPL editor

An ADAPL editor is part of Adlib Designer.

1. In the *Application browser*, open your Adlib folder and in it the subfolder with the ADAPL source files (.ada and .inc files). This folder could be named \adapl sources or \adapls.
2. Click the adapl you want to view and/or edit. The file opens in the upper right window pane.



3. You can adjust the code, if you wish, and recompile or save the adapl: right-click the window pane which holds the code and in the pop-up menu that appears choose either *Compile*, *Compile with debug mode* or *Save*. The results of compiling will be presented in the lower right window pane. Note that the compiled file (.bin) will automatically be saved in the current folder. If you want to test your application with the adjusted adapl, then you may have to move the newly compiled file to the proper Adlib subfolders (which still hold the old version of that .bin file) first; an adapl may occur in more than one folder. (From model application version 4.2 though, the source files and compiled files will be located in one and the same folder.)

Colour coding

To improve the readability of the code, colour coding is automatically applied:

- Blue: reserved words in the ADAPL programming language;
- Red: strings (fixed text values in between quotes);
- Green: comments.

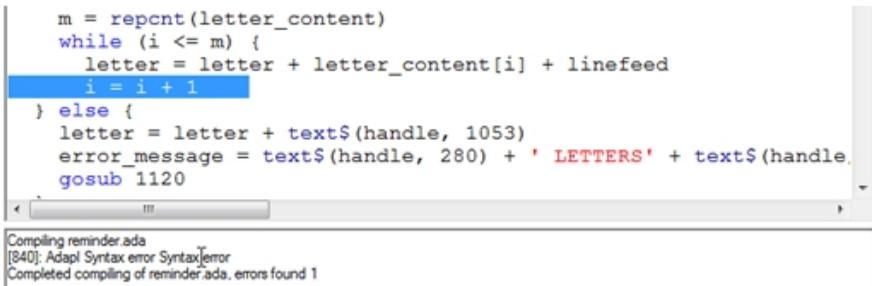
Finding matching braces

In ADAPL code a lot of code is nested and grouped via braces: {...}. In complex pieces of code it may be hard to find the opening brace for a closing brace, or vice versa. In the ADAPL editor it is therefore possible to locate an accompanying brace via a shortcut:

1. Put the cursor directly to the left of the brace for which you want to find the matching one, or select that brace.
2. If you are at a closing brace, then press **Ctrl+←** (**Ctrl** and left arrow key) to locate the opening brace.
If you are at an opening brace, then press **Ctrl+→** to locate the closing brace.
The cursor will automatically be moved to the matching brace.

Finding compilation errors

If errors are found during compilation, you can highlight their location by double-clicking the relevant error message in the compilation report. In the example below this would be the error message line that starts with [840]. In the code, the line where something is wrong will then be selected for you (here, a closing brace is missing).



```
m = repcnt(letter_content)
while (i <= m) {
  letter = letter + letter_content[i] + linefeed
  i = i + 1
} else {
  letter = letter + text$(handle, 1053)
  error_message = text$(handle, 280) + ' LETTERS' + text$(handle,
  gosub 1120
```

Compiling reminder.ada
[840]: Adapl Syntax error SyntaxError
Completed compiling of reminder.ada, errors found 1

5.1.3 Compiling an adapl

An *.ada* text file must be compiled by the ADAPL compiler *adapl.exe* or by Designer or by the ADAPL Debugger to create an executable program.

Compiling with Adlib Designer

An opened *.ada* text file in Designer can be compiled normally either by pressing **F6** or by right-clicking the ADAPL code and choosing *Compile* in the pop-up menu. You can also compile it in debug mode (which is required for the ADAPL Debugger*) by pressing **Ctrl+F6** or by right-clicking the ADAPL code and choosing *Compile with debug mode* in the pop-up menu. The compiled file will be placed in same folder as the ADAPL source code file.

Compiling with *adapl.exe*

Compiling with *adapl.exe* needs to be done from the (DOS) command prompt**, so you'll have to open a command-line window first via the Windows *Start* menu. Using standard DOS commands in this window, go to the directory that holds the *.ada* text file that you want to compile. Now call *adapl.exe*, which is probably in your Adlib *\bin*, *\executables* or *\tools* directory, and use the *.ada* file name as the main argument. You can use a full or relative path to *adapl.exe*. To compile the *notify.ada* file, use for example:

```
"C:\adlib software\executables\adapl" notify.ada
```

or

```
..\executables\adapl notify.ada
```

The basic syntax for starting the ADAPL compiler is:

```
adapl [arguments] filename
```

With the arguments you can indicate that something extra must happen during compilation. An argument consists of a hyphen and one or more characters. Entering an argument is optional. In most cases, the compiler will compile the text file perfectly well without arguments. For the file name fill in the source *.ada* text file. You can leave out the extension, because the file to be compiled cannot have any other extension. The simplest form of a compile command will look something like this if both *adapl.exe* and *test.ada* are in the same directory:

```
adapl test.ada
```

If compiling was successful, the compiler will give a message similar to the following:

[ADAPL 6.6.0.0 (x86), Built at Oct 17 2011,10:38:33.]
 [Copyright (c) 1984-2014, Adlib Information Systems]
 [0 errors and 0 warnings in 311 lines (..\[path]\[filename])]

If there were errors during compilation, no *.bin* file will have been created. You'll will have to re-examine your code, fix the error(s) and recompile the *adapl*.

If there were only warnings, the *adapl* will have been compiled correctly: warnings often warn against declared variables which you never use, for example, but those are not errors. Another warning is of the type: *WARNING: at line 34 of borrstor.ada, unable to read dictionary field list COPIES, error 16*. This is also just a warning, not an error, which occurs if you compile an *adapl* in a folder from where the compiler is not able to locate your *\data* folder by means of your FACS declarations: in the FACS declarations in your *adapl*, the path to the *\data* folder is usually relative, like `fdstart BORROWER '..\data +borrower'`. This is not a problem if you've only used tags in your FACS declarations, like you normally do: it just means that the compiler was not able to check your declarations against the *.inf* files in said *\data* folder.

The compiled *adapl* will have been created in the directory where you currently are. In current Adlib model applications that is usually exactly the place where it should be, but older or custom applications often have compiled *adapls* in other Adlib folders. Sometimes even, copies of the same compiled *adapl* reside in multiple Adlib folders. If your *adapl* didn't exist before, you can place it in a suitable folder of your choice, but if you updated an existing *adapl*, you must check all your Adlib (sub)folders for the older version of the *adapl* and overwrite it with the new version; all of the older versions must of course be identical, otherwise you may lose changes in a deviating older version.

The following command-line arguments can be used with *adapl.exe*:

Argument	Name	Description
-d	Debug	The compiler prepares the <i>.bin</i> file for debugging. Only <i>adapls</i> compiled in debug mode will automatically launch in the ADAPL Debugger* when the <i>adapl</i> is executed by the Adlib application, by <i>import.exe</i> or by Designer's import functionality (the latter available from Designer version 7.1). When, after debugging, your <i>adapl</i> runs fine, remember to recompile it without the -d option, so that executing the <i>adapl</i> won't open the ADAPL Debugger anymore.

-h	Help	The compiler displays a Help text.
-t	Title	The compiler prompts for an ADAPL program title. You can enter one title for each language: <code>title[x]='title'</code> , with <code>x</code> representing the language number.
-v	Verbose	The compiler gives progress information during compilation.

It is possible to enter more than one argument, for example :

```
adapl -dtv test.ada
```

If a single source file name is entered, this file will be compiled by *adapl.exe*. If no errors are found, the resulting output will be written to a file with the same name, but with the extension *.bin* . This *.bin* file will contain the executable form of the ADAPL program. If an error occurs during compilation, a message will be displayed stating the type of error and the line in which it occurs.

You may also compile multiple files at once, and wildcards are supported. The compiler will compile all given files separately and write each of them to the current directory as *.bin* files. For example: the `adapl *.ada` command compiles all files with the extension *.ada* into executable files with the same name but with the extension *.bin*.

If the `-t` argument is used, the compiler will prompt in advance for a short title for the program, which will be displayed in a pick list. If wildcards and the `-t` option are used in combination with each other, the title will be used for all executable programs. It is therefore better to use a `TITLE` instruction in each ADAPL source file, if you want to use it.

Note that the current version of the ADAPL compiler generates 32-bit code, which allows for binary ADAPL code files of up to 2 GB (instead of the 64 KB files that were allowed with the 16-bit compiler) so that you can write much longer ADAPL routines, and this also means that long file names are supported.

An `adapl` compiled with the 32-bit compiler will not work in `adlwin` versions older than 4.5.

* The ADAPL Debugger provides a programmer-friendly way to debug `adapls`. See the ADAPL Debugger Help file for more information about this tool.

** From Windows Explorer you can also compile `adapls` by right-clicking an *.ada* file and choosing *Open with... > ADAPL Compiler* in the pop-up menu; if *ADAPL Compiler* is not present in this menu, you'll have look for *adapl.exe* on your system first. However, compiling an

adapl this way does not allow you to enter command-line arguments, so you can't compile an adapl in debug mode this way.

5.1.4 Using ADAPL programs

ADAPL is a complete programming language which can access every part of Adlib databases and applications. ADAPL programs can be written to perform certain tasks independently of the application itself, e.g. to print reminders for overdue books.

ADAPL programs can run independently using the `adeval` executable, they can run in the background after the user has performed certain actions in an Adlib application, or may be included in the database list of step 1 in the *Search wizard* of an Adlib application.

The Adlib record interface is what makes this language unique and very efficient. Namely, an adapl you use from within Adlib (after you've opened a data source) is always executed per record; depending on where in the execution of Adlib the adapl is started - see below, this could mean just the currently opened record, or all marked records in a brief display for example. Each time the adapl is being run, a record is copied into a buffer (called `adlib_lite`). It is this buffer from which the adapl can subtract all data immediately by using the record tags as if they were variables, and edit them if necessary. The edited tags in the buffer will be copied back automatically to the actual database record only if the adapl is of a type that automatically writes back from the buffer, namely a before-storage adapl, that is carried out when the user saves or deletes a record, or when importing records. Most other types of adapls write any edited tags automatically back to a record, if it is in edit mode, but these are only saved in the record if the user approves and saves the record manually; these are before- and after-screen adapls, field based procedures, and copy, input, and edit record procedures. In the remaining types of adapl you can only save edited tags by using FACS for the local database (see FACS).

Screen procedures

Before-screen adapl - Before-screen adapls (also known as before-display procedures) can be linked to a brief display screen or a detailed display screen through the *Screen* properties of the form. The corresponding adapl is then run immediately prior to any linked screen being opened for each record that is presented. The ADAPL program accesses the data from the record as database variables and is therefore able to edit, suppress or otherwise process the presented data.

One practical use of a before-screen adapl, is to format certain field

entries and add punctuation marks or to swap first names and surnames so that they read naturally. You can also ask for input from the user before the data is displayed. This could be used, for example, to display confidential information only after an access code has been entered.

After-screen adapl - After-screen adapls (also known as after-display procedures) can be linked to a brief display screen or a detailed display screen through the *Screen* properties of the form. The corresponding adapl is then executed immediately after the screen is closed, i.e. when you save a record or enter another screen. This can be used, for example, to run validation checks on data that has been entered or to copy certain data to other Adlib files. The ADAPL program can also be used to force the user to add or edit record data before the record can be saved. The ADAPL program will be run every time the screen is left.

Database procedures

On the *Procedures* tab of the database properties it is possible to link six types of ADAPL programs to a database definition, namely: *Storage procedures*, *After retrieval procedures*, *Copy record procedures*, *Input record procedures*, *Edit record procedures*, and *Field based procedures*. These adapls are executed automatically after a user performs a certain action on this database in an Adlib application. See the Help topic for the *Procedures* properties tab, for more information about each adapl type.

Note that adapls linked to screens take precedence over those specified for databases.

Database field procedures

Link screen procedure (filter adapl) - This adapl is used in link screens to determine which records are displayed or not displayed in this screen. It can also be used to decide which elements of a record you want to display. You can set which *Link screen procedure* you want to use, in the *Link screens* properties of a data dictionary field.

Manually started procedures

- *Print adapl* (stand-alone or output format) - Print adapls can either be stand-alone adapls (adapls which can be started from step 1 in the *Search wizard* of Adlib or outside of Adlib) or output formats associated with a data source. Output formats are executed for every record marked for printing, while the stand-alone adapls must have built-in routines to search databases for certain records and print them; therefore their construction differs significantly.

For some details about how to use ADAPL for formatting plain text printer output, please see the Help topic Output formats.

- *Sort adapl* - This adapl is used to pre-process data from selected records. Sort adapls do not actually sort records. You can use them to fill a tag with a compound value for example, and use that tag to actually sort on by entering it in the appropriate sort option in Designer or in adlwin's sort functionality in the *Expert search language*, the *Brief display* or *Pointer files* window.
- *Import adapl* - An import adapl does not import records itself: per record which is being imported, it processes data from the exchange file to allow for some data conversion before it is actually written to the target fields. An import adapl can be set in import jobs: it runs during the actual import.

5.1.5 Language basics

An ADAPL source code file consists of statements with a maximum length of 1000 ASCII characters per line. A line may start with a number, followed by an ADAPL instruction, and then comments.

Statements which start with * (only when at the first position in the line) and text following /* (may be preceded by spaces, tabs or ADAPL code) are regarded as comments and will be ignored by the compiler. Comments start and end on the same line. You can use comments to include additional details in the program text about what the program is supposed to do.

You can use tabs, blanks and empty lines wherever you wish to ensure that the source code is readable.

Upper and lower case are considered equal in instructions and functions, but not in strings and variable names: the variable *Counter* is not the same as the variable *counter*.

ADAPL source code can be divided between various files. In that case, the compiler will temporarily go to another file during compilation, and go back to compiling the first file as soon as that file has been compiled. The command used for this is `include`.

5.1.6 Reserved words

The names of ADAPL instructions and functions may not be used as variable names, regardless of whether upper or lower case letters are used. Most punctuation marks are reserved for certain functions in ADAPL program texts. You may not use them for any purpose other than their intended purpose within ADAPL. That also means that they cannot be used in names of variables and database fields:

() { } [] < > ! & = ^ * / + - ' , "

The underscore `_` and the percent sign `%` can be used freely.

The table below contains a list of reserved words. The list comprises functions and instructions, internal variable or constant names, and words reserved for future use:

abs	and	after\$
asc	attr	auxport
before\$	black	blinking
blue	box	break
bright	case	chr\$
clear	close	closeall
closeallfiles	closefile	clr
cls	column	continue
cursor	cvt\$\$	cyan
datdif	date\$	dayofweek
default	delete	dialog
dim	display	do
edit\$	else	
end	errorf	errorm
fdend	fdstart	file
form	getkey	getvar

gosub	goto	green
hidden	if	include
input	instr\$	int
integer	is	isin
isisbn	isissn	ismodified
isolatintohtml\$	isstopword	jstr\$
last	left\$	len
let	line	locate
lock	magenta	max
message	mid\$	milestone
min	mod	name\$
ndays	next	no
noexec	nonmessage	noreset
not	null	numeric
off	on	onchangein
oneoj	oneop	onscreen
onsoj	onsop	open
openfile	or	output
page	pagebreak	pdest
pointer	prescan	previous
primos	print	progress
quit	read	reccopy
redate\$	red	redisplay
repcnt	repcopy	repfind

repins	repmax	repmin
repsort	repsortins	repsum
return	reverse	right\$
rinstr\$	round	round\$
sendmail	screen	select
setfont	setlinespacing	setorientation
setpapersize	setstatusbartext	setvar
setwindowtitle	show	skip
spool	sqrt	status
str\$	string\$	switch
system	tag2field\$	term\$
text	text\$	then
time\$	title	trim\$
unlock	until	update
user\$	using	val
whatdate	while	white
write	writeempty	xor
yellow	yes	yesno

5.1.7 Data types

ADAPL allows three different data types:

- **integer:** whole decimal numbers (e.g. 10, -350)
- **numeric:** decimal numbers (e.g. 3.5, -E30)
- **text:** character strings (e.g. 'Word', 'a')

The ranges of these data types depend on the host operating

system. Under MS-DOS, the following values apply:

integer	Ranges from -2147483648 up to and including +2147483647.
numeric	Ranges from -1.7*10308 (-17 followed by 307 zeros) to +1.7*10308.
text	Can hold up to 65000 bytes, but never more than the number of bytes reserved for the text variable, like so for instance: <code>text MyText[1000]</code> Note that under DOS, one character is represented by one byte, but in Unicode UTF-8 character sets under Windows, letters with a diacritical character and exotic letters are usually represented by two (or sometimes three) bytes. So declare your text variables at least twice the length of the number of characters they're supposed to hold, if a variable is going to hold data from records.

You will often be saving these data as variables. A variable is simply a piece of memory to which you have given a name, making it easy to manipulate its content. You can give assignments to ADAPL such as 'put a value in the piece of memory with this name'. At a later stage, you can change the value, or use it in a calculation. When you indicate that you want to use a variable, you also indicate how much (maximum) and what type of information you want to save in it. It is not necessary to state to which piece of memory you want to refer - Adlib does that for you.

You may also want to refer literally to a value. For example, if you want to use the number '10', you key that number in literally. We call this kind of literal value a constant. Whereas a variable may regularly change in value, the value of a constant stays the same.

The data type of a constant is determined by a notation and by the context in which it occurs. The data type of a variable is determined by a declaration. In an ADAPL program, you have to indicate the data type of a variable in advance.

If necessary, Adlib will automatically convert variable values and results of instructions to a data type with fewer restrictions, in the order INTEGER > NUMERIC > TEXT. To convert variables in the other direction you can use the functions `VAL()` and `INT()`.

An expression is an arithmetical expression such as `a=b+10`. The data type of the result of an expression depends on the data types of the

parts of the expression. If different data types are used in an expression, the result will assume the least restrictive type.

In contrast with a numeric, an integer is a whole number. That makes an integer particularly suitable for counters and numbers. Numerics are not 100% exact, but do have a certain precision. The bigger the number, the less the precision. A numeric can therefore handle bigger numbers than an integer.

See also

Declaring local variables

Constants and variables

5.1.8 Constants

In an ADAPL program, a constant is a value that is included literally in the program text. In an expression such as `a=b+10`, `10` is a constant. Whatever happens in the program, a constant always keeps the same value. In contrast, the value of a variable may change. ADAPL allows three types of constants, corresponding with the three data types: integer, numeric, and text.

ADAPL will recognize a string of digits without a decimal point as an integer constant. An integer constant may be preceded by a minus sign. If an integer notation occurs in a situation where a numeric is expected, the result will be a numeric constant. For example, in the instruction `digit = 2`, where `digit` is declared as a numeric value, `2` will be considered a numeric constant.

A series of digits with a decimal point and/or an exponent character (`E` or `e`) is seen by ADAPL as a numeric constant. The decimal notation may be preceded by a minus sign. With an E-notation (scientific notation) the decimal part (the mantissa) and the exponent may both be preceded by a minus sign. ADAPL treats numeric constants as floating point numbers.

A series of characters embedded in single quotes is treated as a text constant by ADAPL. If nothing is typed between the quotes, what you have is an empty text, also called an empty string or a null string. This is particularly useful for initializing variables of the text type.

All characters in a particular character set can be included as part of a text constant. That includes accented characters, non-visible signs and reserved signs such as `'/'` and `'*'`. Single quotes and backward slashes can also be incorporated into a string, but then they must be

doubled (this is called "escaping" of reserved characters). For example:

```
DISPLAY 'He said: ''Hello!''.'
```

* The literal quotes are doubled.

```
DISPLAY 'The path to Adlib is: C:\\Apps\\Adlib Software'
```

* The literal slashes (\\) are doubled.

When these instructions are executed, the following will be displayed:

```
He said: 'Hello!'.  
The path to Adlib is: C:\\Applications\\Adlib Software
```

One single quote can be displayed by using four consecutive quotes (see also Text expressions):

```
DISPLAY ''''
```

* The outer two quotes serve to denote a string is in
* between them. Of the inner two quotes the first marks the
* second to be interpreted literally; only the third quote
* of the four, will be shown.

5.1.9 Variables

Introduction

Adlib allows three main types of variables: local variables, database variables and reserved variables.

- **Local variables** are only available within an ADAPL program.
- **Database variables** refer not to a piece of memory but to a field in a database.
- **Reserved variables** are variables that have been predefined by the developers. These variables can be retrieved, and sometimes filled, but their meaning cannot be changed.

All variables are available anywhere in an ADAPL program and retain their value in all subroutines and instruction blocks.

In addition, FACS variables can be declared for the FACS subsystem. As far as naming is concerned, these are treated as local variables, but have the characteristics of database variables.

Names

Local variables

Names of local variables can be from 1 up to 32 characters long. They consist of a series of letters and digits and may include underscores ('_'), but no other punctuation marks. The first character of a variable name must be an underscore or a letter, and may therefore not be numeric. ADAPL is case-sensitive. The variable `Counter` is therefore not the same as the variable `counter`. Local variables are defined by the ADAPL programmer using declaration instructions.

Database variables

Names of database variables consist of two characters, of which the first is a letter or a '%' sign, and the second a letter or a digit. These variables are commonly known as 'tags'. Database variables do not have to be declared, but automatically become available if the ADAPL program is executed within the Adlib application for a certain record. If the `adapl` is executed outside of an Adlib application or from within step 1 in the *Search wizard* of an Adlib application (when no database has been opened yet), you'll have to use FACS to access database variables.

Adlib considers all non-declared variables of 2 characters as database variables. Database variables may have been defined in the data dictionary of a database, but that is not a requirement.

Any value obtained from a database variable is considered a string. This is relevant for value comparisons, assigning tag values to local variables, numeric functions and arithmetic expressions. Therefore, values obtained from database tags which you want to treat as numbers, must always first be converted from strings to numerics (VAL) and then possibly to integers (INT) before they can be compared as numbers or be used in calculations.

A database tag may have up to 32.767 occurrences. A field occurrence is repetition of the field, containing a different value. So, if for instance the Author field is a repeatable field, then the user can enter more than one author in this field, each author in its own field occurrence. In ADAPL, you can address specific field occurrences using the syntax: `<tag>[<occ>]`, so to address the second occurrence of the Author field, we would write: `au[2]`. Occurrence numbers are 1-based, meaning that the first occurrences has the number 1. In Adlib SQL and Adlib Oracle databases, fields can be set up to be multilingual. This means that every field occurrence may contain values in multiple translations. (The user only sees one translation at a time, depending on the currently chosen data language.) From version 6.6.0, such multilingual field occurrences can be addressed (for reading and writing of data) in ADAPL. The syntax for addressing

one particular language value in a field occurrence is: `<tag>[<occ>, 'language code']`, for example: `te[1, 'en-US']` to address the English-US translation of the first occurrence of a multilingual Term field. (The language code string may be a text variable as well.) The occurrence number is mandatory. An example of a value assignment would be: `te[1, 'en-US'] = te[1, 'en-GB']`. If you also want to make the target language the invariant language, you can do so by inserting a hash character in front of the target language code, for example: `BA[1, '#fr-FR'] = BA[1, 'fr-FR']` to set the French language value as the invariant one. If, when reading, the desired occurrence is empty whilst the same occurrence does have a value in a different language that happens to be the invariant value, then that value is retrieved instead.

Reserved variables

Reserved variables are reserved tags that start with '&'. They are not declared.

The information in reserved tags is passed on to ADAPL by Adlib. The table below contains a list of reserved tags and the information they provide.

Tag	Type	Information
&0	numeric	Number of the current record. If you use this variable in a derival procedure (such an adapl is executed before and after derival), then before derival &0 will contain the record number of the source record, while after derival it will contain the record number of the target record (it this is an existing record). If you started to derive a source record into a new record (not an existing one) then after derival &0 will contain the number 0; the new record has not been stored yet so no actual record number is available.
&0[2]	numeric	Number of the current source record in case this variable is used in a derival procedure. If you use this variable in a derival procedure (such an adapl is executed before and after derival), then before and after derival &0[2] will contain the record number of the source record. (This occurrence of &0 is available from 6.6.0.550.)

&0 [3]	numeric	Number of the current target record in case this variable is used in a derival procedure. If you use this variable in a derival procedure (such an adapl is executed before and after derival), then before and after derival &0 [3] will contain the record number of the target record (it this is an existing record). If you started to derive a source record into a new record (not an existing one) then before and after derival &0 [3] will contain the number 0; the new record has not been stored yet so no actual record number is available. (This occurrence of &0 is available from 6.6.0.550.)
&1	numeric	Execution code (See next table: 'Execution codes &1')
&1 [2]	numeric	Execution sub codes (See the 'Execution sub codes &1 [2]' table below). This reserved variable provides more information about the circumstances under which certain execution codes (&1) were generated.
&4 [1]	text[8]	Name of current screen
&4 [2]	text[2]	Current tag
&4 [3]	text[10]	Number of current occurrence as text
&5	numeric	Adloan.exe current transaction code (See table: 'Execution codes &5')
&6 [1]	text[80]	Name of database in which the main record is located that the user has opened. If, from within some main record, the user opens a zoom screen to a record in another database, then this zoomed record becomes the new main record that the user has opened.
&6 [2]	text[80]	Name of dataset in which the main record is located that the user has opened. If, from within some main record, the user opens a zoom screen to a record in another database, then this zoomed record becomes the new main record that the user has opened. Names are case-sensitive.

&6[3]	text[80]	<p>Path name of the dataset in the database in which the main record is located that the user has opened.</p> <p>If, from within some main record, the user opens a zoom screen to a record in another database, then this zoomed record becomes the new main record that the user has opened.</p>
		<p>A problem with the above described three variables is that when during editing or saving of the main record another record is implicitly created or updated in a linked database (through a linked field in the main record or adapl), said variables still contain values related to the main record.</p> <p>So if you would like to call up properties of the database in which the system is working when some record not explicitly opened by the user is being updated or created, to use, for instance, the name of the linked database to fill an automatic field or as a condition for executing certain adapl code, you need the three variables described below.</p>
&6[4]	text[80]	Name of database that the system has opened.
&6[5]	text[80]	Name of dataset that the system has opened. Names are case-sensitive.
&6[6]	text[80]	Path name of the dataset in the database that the system has opened.
&B[1]	numeric	Number of records in selection (after a limit, if applicable).
&B[2]	numeric	Total number of hits (before any limit is imposed).
&B[3]	numeric	This variable can only be used with wwwopac.exe. See the wwwopac reference guide.
&B[4]	numeric	This variable can only be used with wwwopac.exe. See the wwwopac reference guide.
&B[5]	numeric	This variable can only be used with wwwopac.exe. See the wwwopac reference guide.

&C[1]	text[64]	Index value used to retrieve record
&C[2]	text[80]	End key
&C[3]	text[80]	Free text key
&C[4]	text[80]	Name of the domain from which the key was retrieved.
&C[5]	text[80]	Index value with which a FACS READ found a record. This can be useful when reading an index using wildcards or truncation, and you want to know the currently retrieved index value.
&D	numeric	Index tag used to retrieve record
&E	numeric	FACS result code; Exit code
&F	text[2]	Gives tag of access point used
&G	numeric	Number of access point in Adlib
&H[n]	text[n]	Contains command line arguments. &H makes it possible to invoke the program with one or more arguments. (adeval adaplname argument) &H[1] is filled with the name of the adapl, &H[2] with the first argument, &H[3] with the second argument, etc.
&I	numeric	In a print adapl, the serial number (not the record number) of the currently printed record in the selection of marked records.
&L	numeric	In a print adapl, the line number of the next line to be printed in the printout.
&P	numeric	Number of active language (0 is default, 1 is language1, etc.)
&S	numeric	Page count in print adapl
&T[1]	text	Date on which record is edited
&T[2]	text	Time at which record is edited
&T[3]	text	Date on which record is created

&T[4]	text	Time at which record is created
&V	text	During the execution of an update import (the <i>Update tag</i> option in the import job must have been filled), this variable will contain the record number of any currently updated record. On the other hand, if from the exchange file a record is being processed which is currently being added to the database as a new record (because there's no existing record to update), then the variable will be empty. You can use the variable in an adapl associated with the import job, to distinguish between updated records and new records.

ADAPL execution codes that can occur in the reserved variable &1:

Code	Meaning
0	(Reserved)
1	<i>Before screen adapl</i> called before a record is displayed, either in Display, Read-only or Edit mode.
2	<i>After screen adapl</i> called after editing of a record: when leaving a screen (tab sheet) for an existing record (prirf is not zero) in edit mode. Leaving a screen happens when you click another tab, or when you store the record; in the latter case the code first becomes 2 and then 11.
3	<i>After screen adapl</i> called after input of a new record: when leaving a screen (tab sheet) for a new record in edit mode. Leaving a screen happens when you click another tab, or when you store the record; in the latter case the code first becomes 3 and then 11.
4	<i>Before screen adapl</i> or <i>Edit record procedure</i> called before editing a record. See &1[2] to determine which one was called.
5	(Reserved)
6	(Reserved)
7	Adapl executed as stand-alone Adapl with ADEVAL.

8	Adapl called from print function (output adapl).
9	<i>Before screen adapl</i> or <i>Input record procedure</i> called before input of a new record. See &1[2] to determine which one was called.
10	(Reserved)
11	<i>Storage procedure</i> called before storage of a record.
12	<i>Storage procedure</i> called before deletion of a record.
13	Adapl called from the step 1 in the <i>Search wizard</i> .
14	<i>Storage procedure</i> called after storage of a record.
15	<i>Storage procedure</i> called after deletion of a record.
16	<i>Before screen adapl</i> or <i>Copy record procedure</i> called after copying of a record. See &1[2] to determine which one was called.
17	Adapl called before sorting of a record.
18	Adapl called with the F12 function key in display mode (F12Adapl): this execution code is no longer used in Adlib Windows applications.
19	Adapl called with the F12 function key in edit mode (linkproc): this execution code is no longer used in Adlib Windows applications.
20	<i>Before screen adapl</i> or <i>Field based procedure</i> called before display of a field: when entering a field in edit mode of record. See &1[2] to determine which one was called.
21	<i>Before screen adapl</i> or <i>Field based procedure</i> called after input of a field: when leaving a field in edit mode, irrespective of whether the contents of the field have been changed. See &1[2] to determine which one was called.
23	<i>After retrieval procedure</i> called after a record has been retrieved. This may happen in a multitude of circumstances, not only for the detailed display of a record for instance.
24	<i>After derival procedure</i> (ignore the <i>After</i> part) called after deriving a record. To avoid this procedure from being

	executed twice (before and after derival), check the execution sub code before executing adapl program code.
26	<i>After derival procedure</i> (ignore the <i>After</i> part) called before deriving a record. To avoid this procedure from being executed twice (before and after derival), check the execution sub code before executing adapl program code.

Some execution codes from the list above are accompanied by an execution sub code in &1[2] to further specify the circumstances under which the currently executed adapl was called. ADAPL execution sub codes that can occur in the reserved variable &1[2] are:

&1[2] code	&1 codes	Meaning
1	2, 3	This code is produced if &1=2 or &1=3 and the currently executed adapl has been called as an <i>After screen adapl</i> , except after switching tabs.
2	2, 3	This code is produced if &1=2 or &1=3 and the currently executed adapl has been called as an <i>After screen adapl</i> after switching tabs (=screens).
3	1	A brief presentation screen is about to be displayed. This code is produced if &1=1 and the currently executed adapl has been called as a <i>Before screen adapl</i> for a detail or brief screen.
4	4, 9	This code is produced if &1=4 and the currently executed adapl has been called as an <i>Edit record procedure</i> in the current database, or if &1=9 and the currently executed adapl has been called as an <i>Input record procedure</i> in the current database.
5	4, 9	This code is produced if &1=4 and the currently executed adapl has been called as a <i>Before screen adapl</i> while entering Edit mode, or if &1=9 and the currently executed adapl has been called as a <i>Before screen adapl</i> .
6	1	The screen is opened in Edit mode. This code is produced if &1=1 and the currently executed adapl has been called as a <i>Before screen adapl</i> for a detail screen, a zoom screen or a QBF screen.

7	1	The screen is opened in Edit or Display mode*. This code is produced if &1=1 and the currently executed adapl has been called as a <i>Before screen adapl</i> for a detail screen. * Unfortunately this code is also generated in some other cases like from a skipped brief screen (with only one record on it), after saving a derived record, or before zoom display of a record to be derived. Therefore it's advised to perform no actions at all, if this execution sub code occurs.
8	1, 4	The screen is opened in Display mode. This code is produced if &1=1 and the currently executed adapl has been called as a <i>Before screen adapl</i> for a detail screen, a zoom screen or a QBF screen, or if &1=4 and the currently executed adapl has been called as a <i>Before screen adapl</i> while the current mode is Edit mode already or when the screen is opened as a zoom screen.
9	16, 20, 21	This code is produced if &1=16 or &1=20 or &1=21 and the currently executed adapl has been called as a <i>Before screen adapl</i> .
10	16, 20, 21	This code is produced if &1=20 or &1=21 and the currently executed adapl has been called as a <i>Field based procedure</i> in the current database, or if &1=16 and the currently executed adapl has been called as a <i>Copy record procedure</i> .
11	3	This code is produced if &1=3 and the currently executed adapl has been called as an <i>After screen adapl</i> after leaving a QBF screen.

ADAPL execution codes that can occur in the reserved variable &5:

&5 code	Transaction type
92	Issue
93	Return
94	Renewal

95	Payment
96	Reservation

To further clarify the execution sub codes, the table below lists the execution codes first and secondly the execution sub codes that may appear for them.

&1 code	&1[2] codes	Meaning
1	3, 6, 7, 8	<i>Before screen adapl</i> called before a record is displayed, either in detail, zoom or QBF screen in Display (8 or 7), Read-only or Edit (6 or 7) mode, and before display of a brief screen (3).
2	1, 2	<i>After screen adapl</i> called after editing of a record: when leaving a screen (tab sheet) for an existing record (priref is not zero) in edit mode. Leaving a screen happens when you click another tab (2), or when you store the record (1); in the latter case the execution code first becomes 2 and then 11.
3	1, 2 11	<i>After screen adapl</i> called after input of a new record: when leaving a screen (tab sheet) for a new record in edit mode. Leaving a screen happens when you click another tab (2), after leaving a QBF screen (11), or when you store the record (1); in the latter case the execution code first becomes 3 and then 11.
4	4, 5, 8	<i>Before screen adapl</i> (5) or <i>Edit record procedure</i> (4) called when entering Edit mode of a record, or a <i>Before screen adapl</i> (8) while the current mode is Edit mode already or when the screen is opened as a zoom screen.
9	4, 5	<i>Before screen adapl</i> (5) or <i>Input record procedure</i> (4) called before input of a new

		record.
16	9, 10	<i>Before screen adapl (9) or Copy record procedure (10) called after copying of a record.</i>
20	9, 10	<i>Before screen adapl (9) or Field based procedure (10) called before display of a field: when entering a field in edit mode of record.</i>
21	9, 10	<i>Before screen adapl (9) or Field based procedure (10) called after input of a field: when leaving a field in edit mode, irrespective of whether the contents of the field have been changed.</i>

Initialization

Before you can use a variable, Adlib must initialize it. That means Adlib will reserve memory for it and assign it a starting value. This takes place automatically. Local variables are initialized when the program is loaded, and retain their value until program termination or until a different value is assigned. Database variables are available automatically if the adapl is running within Adlib. As soon as a new record is loaded, the variables assume the values of the fields in the new record.

If an ADAPL program is executed for a number of records consecutively, the declared variables will take their value with them from the first execution to the second and subsequent executions, unless explicitly cleared. Database variables, on the other hand, will be reinitialized and will take their values from each record in turn as it is processed.

FACS variables are initialized when the corresponding external database is opened. When a new FACS record is read, the variables assume the values of the fields in the new record.

Arrays

Normally speaking, it is only possible to store one datum in a local variable. However, you can specify that several data must fit in one variable by referring to the various data with sequence numbers, placed after the variable name in square brackets. A multiple variable like this is known as an array. A datum in an array can in turn consist

of a number of data. In that case, it will be an array with several dimensions. A declaration of a text array of 5 times 2 elements with a maximum length of 20 characters will look like this:

```
text textarray[5, 2, 20]
```

The subscripts (the numbers between square brackets) in the declaration are integer constants, which define the number and size of the dimensions in multiple variables. An array can have a maximum of 4 dimensions and may take up a maximum of 64 KB of memory. A text variable is handled as an array of characters, so the last dimension of the subscripts indicates the length of the string. In text expressions, the last dimension subscript may be used to address an individual character within the string. Thus, the following function call returns the third character of the second column in the first row of a text array:

```
character = textarray[1, 2, 3]
```

Example of an ADAPL in which arrays are used:

```
text textarray[5, 2, 20]
  /* text array of 5*2 elements, length 20
integer i /* array pointer
* fill the array
i = 1
while (i <= 5) {
  textarray[i, 1] = 'cell ' + i + ',1'
  textarray[i, 2] = 'cell ' + i + ',2'
  i = i + 1
}

* read the contents of the array and
* display the contents on screen

cls /* clear screen
i = 1
while (i <= 5) {
  display '' /* new-line
  display textarray[i, 1] + ' | ' + textarray[i, 2]
  i = i + 1
}
display 'enter to continue...'
input(0)
end
```

The result:

```
cell 1,1 | cell 1,2
```

```
cell 2,1 | cell 2,2
cell 3,1 | cell 3,2
cell 4,1 | cell 4,2
cell 5,1 | cell 5,2
Press enter to continue...
```

This example shows that a two-dimensional array can be seen as a table. The first subscript represents the row, and the second the column. Together, they constitute the unique identification of each cell in the table.

5.1.10 Expressions

An expression is a constant, a variable, an array element, a function or a combination of these with one or more operators. The result of an expression will be an integer, a numeric or a text value. Operators are discussed below. Expressions can be used wherever constants can be used: in assignments, in comparisons, as part of a larger expression or as an argument in an ADAPL function.

Arithmetic expressions

If a calculation consists of a number of different expressions (e.g. $2 + 3 * 2 / 10 - 5$), Adlib will evaluate them in a fixed order. This order corresponds with the standard arithmetic order: exponentiation, multiplication, division, integer modulus, addition, and subtraction. The table below shows the operators that can be used in numeric expressions, in order of evaluation:

Operator	Description
()	Round brackets (parentheses) enclose an independent expression
^	Exponentiation
*	Multiplication
/	Division (requires numerical values or variables to produce a numerical result)
\	Integer modulus (the remainder of a division)
+	Addition

-	Subtraction
---	-------------

The following functions with a numeric or integer result are available in arithmetic expressions:

abs()	Absolute value of a number
asc()	ASCII value of a character
box()	Draws a box
clr()	Clears part of the screen
datdif()	Number of days between two dates
dayofweek()	Gives the day number for the date (0 is Sunday ... 6 is Saturday)
getkey()	The ASCII or function code of a key
instr\$()	Position of a character in a string
int()	The whole (integer) part of a value
len()	The actual length of a character string
max()	The maximum value of a series of values
min()	The minimum value of a series of values
mod()	The remainder from the division of two values
ndays()	The number of days until (negative value) or since the beginning of 1900 (positive value)
repcnt()	The number of occurrences of a tag
repfind()	Searches for a text string in a tag
repmax()	The maximum value in a tag
repmin()	The minimum value in a tag
repsum()	The total of occurrences of a tag

<code>rinstr\$()</code>	The position of a character in a string
<code>round()</code>	The rounded value of a number
<code>sqrt()</code>	The square root of a number
<code>val()</code>	The numeric value of a character string

Text expressions

One operator is available in ADAPL for text expressions: `+`. The 'plus' sign links, or concatenates, text strings, for example:

```
DISPLAY 'He said:' + ' ' + 'Hello!' + ' ' + '.'
* four consecutive quotes as separate (sub) string are
* needed to show only one quote (see also ADAPL constants).
```

The result of this instruction appears on screen: *He said: 'Hello!'*.

The following functions with a text result are available for text expressions:

<code>after\$()</code>	Returns a substring to the right of indicated character(s)
<code>before\$()</code>	Returns a substring to the left of indicated character(s)
<code>chr\$()</code>	Returns a character from the ASCII table
<code>cvt\$\$()</code>	Converts from/to upper case
<code>date\$()</code>	Returns the system date as a string
<code>getvar()</code>	Reads the contents of a global system variable
<code>input()</code>	Reads a string from the keyboard
<code>isolatintohtml\$()</code>	Converts a string from Isolatin to HTML
<code>jstr\$()</code>	Justifies or centres a line
<code>left\$()</code>	The left part of a character string

<code>len()</code>	The actual length of a character string
<code>mid\$()</code>	Any part of a character string
<code>name\$</code>	Changes 'surname, first name' to 'first name surname'
<code>reccdate\$()</code>	Input or mutation date or time as a character string
<code>right\$()</code>	The right-hand part of a character string
<code>round\$()</code>	The rounded value of a number
<code>str\$()</code>	Converts a value to a character string
<code>string\$()</code>	Generates a string of equal characters
<code>tag2field\$()</code>	Returns the name of the tag as entered in the database setup. If no name is filled in there, the tag will be returned
<code>text\$()</code>	Reads line from text file
<code>time\$()</code>	The system time as a character string
<code>trim\$()</code>	Removes leading and/or trailing blanks
<code>user\$()</code>	User name or user number as character string
<code>whatdate()</code>	Returns a computed date as a character string

Logical expressions

Logical expressions are expressions that contain logical operators and operators for comparison. The result of a logical expression is either TRUE or FALSE. These values are defined numerically: 0 is FALSE, any other value is TRUE.

Integers and numerics are interchangeable. With text comparisons, the value of the logical expression is numeric.

ADAPL allows the following operators for logical expressions:

Operator	Description	Note
()	Round brackets embed an independent expression	
=	Equals.	String comparisons are always case-sensitive.
<	Less than or alphabetic precedence	Number values obtained from database tags must always first be converted to numerics (VAL) and then possibly to integers (INT) before they can be compared as numbers or be used in calculations.
=< or <=	Less than or equal to	
>	Greater than or alphabetic succession	
=> or >=	Greater than or equal to	
<> or ><	Does not equal	
AND	TRUE if both expressions are TRUE	
NOT	Reverses the result of the expression following it	
OR	TRUE if at least one out of two expressions is TRUE	
XOR	TRUE if one expression is TRUE and the other is FALSE	

Tag indirection

Tag indirection means that the content of a text variable is treated as if it were a database tag. Typically, tag indirection is used when the database tags you wish to use in your code, are also ADAPL reserved words or commands, like for instance the `do` tag in the PEOPLE database: only by entering the tag as a string (in between quotes) into the code, ADAPL won't treat it as a command.

The indirection operator, the exclamation mark, is placed immediately before this variable, to use this functionality. To access subsequent elements of the tag, simply add the required occurrence number, enclosed in round or square brackets, to the name of the string.

The following piece of code displays the first line of the title of a book:

```
text tag_var[2]
tag_var = 'ti' /* title
display !tag_var[1]
```

With a dataset that has been opened using FACS, the FACS name of the dataset must be included in the text variable before the tag, e.g.:

```
text domain_tag[15]
domain_tag = 'PEOPLE do'
display !domain_tag[1]
```

This piece of code will display the first occurrence of the domain field in the PEOPLE database.

Note that tags in an opened FACS database addressed this way, do not have to be specified in the FACS definition, because if a database tag you want to use in your code, is also an ADAPL reserved word or command, you simply can't specify the tag in a FACS declaration.

The indirection operator may also appear in a different context, to directly address a tag in a record buffer, including tags which are also reserved words in ADAPL. Records buffers include FACS-declared databases and `adlib_lite` (which contains the contents of a currently edited record). For example, to retrieve the contents of the `do` tag from the currently edited record and display it in an error message:

```
errorm adlib_lite!do
```

Another example for the case in which a FACS database *People* has been declared:

```
if (enumval$(People!do, People!do, -1) = 'AUTHOR'){ ...
```

Sequence of evaluation in an expression

In the case of compound expressions, Adlib will evaluate the various expressions in a fixed sequence. This sequence is based on the standard arithmetic sequence: exponentiation, multiplication, division, addition, and subtraction. Adlib adheres to the following order:

1. tag indirection
2. unary minus (the minus sign of a negative value)
3. functions
4. exponentiation
5. multiplication and division

6. modulus
7. addition and subtraction.

In the event of equal priority, ADAPL will evaluate an expression from left to right.

Expressions and functions can occur within other expressions and functions if they are embedded in round brackets (). The evaluation always begins with the expression in the innermost round brackets, then working outwards. If expressions are used for comparisons, the expressions are evaluated before the comparison is carried out.

The sequence of evaluation in logical expressions is:

1. tag indirection and arithmetic evaluation
2. comparative operators
3. logical operators.

Independent expressions which are embedded in round brackets are evaluated first.

5.1.11 Instructions

ADAPL instructions are simple words used as directions to the compiler. The words are similar to those used in programming languages such as BASIC and PASCAL. They are divided into the following categories:

- Compiler instructions are direct instructions to the ADAPL compiler.
- Declarations define variables and their data type.
- Assignment instructions modify the values of variables.
- Input and output instructions take care of keyboard input, and output to screen, printer or file.
- Control instructions take care of the execution sequence within a program.
- Statements can influence the results of procedures.
- FACS instructions offer access to Adlib databases.
- Functions are for manipulating data.

Compiler instructions

These instructions are executed by the compiler during compilation.

Instruction	Short description
include	'filename' will be compiled, then compilation will continue of the current file.
quit	The compiler stops compiling. If present, errors or warnings will be displayed, but no executable (.bin) file will be generated.
title	Assigns a title to an ADAPL program. When the adapts are displayed on a menu, the title is displayed beside it.

Declarations

With the exception of FACS variables, database variables do not need to be declared. They are implicit and have fixed characteristics.

In ADAPL, a local variable is always declared with one of the following instructions: `INTEGER`, `NUMERIC`, or `TEXT`.

```
INTEGER variable1, variable2[dimensions], ...
```

```
NUMERIC variable1, variable2[dimensions], ...
```

```
TEXT variable1, variable2[dimensions], ...
```

The names of the variables must comply with rules described here. A variable is declared as an array (repeated variable) by adding dimensions. These have the format `[a]`, `[a, b]`, `[a, b, c]` or `[a, b, c, d]`, in which `a`, `b`, `c`, and `d` define the size of the dimensions.

During the declaration of variables, internal memory is set aside for them. This means that the amount of internal memory in your system can limit the number and length of variables used. It is particularly important to take this into account when declaring large arrays.

`TEXT` variables without specific dimensions are always one character large. If a single dimension is given, this will be taken as the maximum allowed length of the text. If several dimensions are given, the last one will be the length of the elements. This system enables you to address each character in the array separately. If the string

you will be storing in a text variable may just as well remain small or get very large, you can specify the length dimension as zero. The memory for the text variable will be assigned dynamically, as required by the data you store in it.

Examples:

<code>text A[20]</code>	Single text variable with length 20
<code>text B[0]</code>	Single text variable with undetermined length
<code>text C[3, 20]</code>	Text array of 3 strings, length 20
<code>text D[3, 0]</code>	Text array of 3 strings, length undetermined
<code>text E[5, 3, 20]</code>	Text array of 5 arrays (15 strings of 20 characters)
<code>integer N</code>	Single integer variable
<code>integer M[5]</code>	Array of 5 integers
<code>numeric O[2, 3, 2, 6]</code>	Numeric array with 4 dimensions (72 elements)

Assignment instructions

Instruction	Short description
<code>let</code>	The assignment instruction assigns the result of an expression to a variable or an array element. The syntax is <code>LET variable = expression</code> . The word <code>LET</code> can be omitted, because the compiler also

	recognizes the combination <code>variable = expression</code> as an assignment instruction.
--	---

Input and output instructions

The following instructions take care of keyboard input, and output to screen, printer or file:

Instruction	Short description
<code>break</code>	on/off: user may/may not break adapl with <code>ctr1-c</code> (default: <code>off</code>).
<code>cls</code>	Clears all characters from the screen; the entire screen takes on the background colour.
<code>display</code>	Displays the corresponding text expression, starting at the current cursor location.
<code>redisplay</code>	Redisplays the screen. Any modifications applied by ADAPL will become visible, after which the user can start entering or modifying data again.
<code>errorf</code>	Moves the cursor to the specified occurrence of the field, and the field for the tag occurrence will blink after execution of the next <code>redisplay</code> .
<code>errorm</code>	Displays the text on the Adlib screen, and asks the user to press Enter to continue.

The following functions enable terminal and keyboard manipulation:

Function	Short description
<code>getkey</code>	Reads a character or function key from the keyboard.
<code>input</code>	Reads a character string from the keyboard.
<code>locate</code>	Moves the cursor.
<code>onscreen</code>	Finds a field for the tag on the screen.

show	Moves the cursor and then prints text with certain visual characteristics.
------	--

Printing

The following instructions take care of printing.

Instruction	Short description
column	Printing in columns.
onsoj	Subroutine that is carried out when the print job is started.
oneoj	Subroutine that is carried out at the end of the print job.
onsop	Subroutine that is carried out at the beginning of a page.
oneop	Subroutine that is carried out at the end of a page.
output	Sends the line built up by print instructions to the print file.
page	Sets the size of a printable area.
pagebreak	Sets the form feed character.
pdest	Sets the destination of the print jobs.
print	Builds up the print line.
printimage	Print an image anywhere on a page.
setfont	Sets font type and point size (Windows).
setlinespacing	Sets the distance between two lines, including the line itself.
setorientation	Sets the printing orientation to portrait or landscape (Windows).
setpapersize	Sets the paper format (Windows).

wordcreatedocument	Export records to a Word template.
wordcreatelabels	Export records to a Word labels template.
wordprintdocument	Export records to a Word template and print the resulting documents.

Control instructions

Control instructions control the order of execution of statements in the program. ADAPL instructions are carried out sequentially unless the program changes the order of execution itself. ADAPL has three ways of doing this: jump instructions and subroutines, conditional execution, and loop control.

Jump instructions and subroutines

Instruction	Short description
end	Execution of the adapl is terminated. Return to the calling program.
goto	Jump to a line with a particular number in the ADAPL program.
gosub	Jump to a line with a particular number in the ADAPL program, but after return jump back to the point from which you jumped.
return	Continue execution at the line following the last encountered gosub.

Conditional execution and loop control

ADAPL incorporates conditional execution using the `IF ... THEN ... ELSE`, `SWITCH ... CASE`, `DO ... UNTIL`, and `WHILE` constructions. During conditional execution, instruction blocks can be used instead of single instructions. An instruction block starts with a left curly bracket ('{') and ends with a right curly bracket ('}'). The left curly bracket of the instruction block must be on the same line as the keyword `THEN`, `ELSE`, `SWITCH`, `CASE`, `DEFAULT`, `WHILE` or `DO`. The right curly bracket must be on the same line as the `ELSE` or `UNTIL` keyword. If no `ELSE` keyword is

used, the right curly bracket can be placed either following the last instruction in the block, or on a new line. If the instruction block consists of a single instruction, you need not use the brackets. The whole construction will then be on one line.

Below, you will find a short description of the various control instructions.

Instruction	Short description
<pre>if ... then ... else ...</pre>	<p>Check that the <code>if</code> condition is complied with, then carry out the <code>then</code> instructions. If the <code>if</code> condition was not complied with, carry out the <code>else</code> instructions. One of the two instruction blocks will therefore be carried out. An <code>else</code> block is not mandatory.</p>
<pre>switch ... case ... default</pre>	<p>Compare each <code>case</code> expression with the <code>switch</code> expression. If a comparison is true, execute the code following the concerning <code>case</code> statement. If all comparisons fail, execute the <code>default</code> code, if present.</p>
<pre>while</pre>	<p>As long as the expression is true, the instruction block will be carried out.</p>
<pre>do ... until</pre>	<p>Carry out the instruction block, then see if the <code>until</code> condition is complied with, and then carry out the instruction block again as long as the expression remains true. The instruction block is therefore carried out at least once.</p>

Statements

select no

What `select no` does in an `adapl` depends on the purpose of the `adapl`. Generally speaking, `select no` can be used as a kind of filter, in various environments:

- **Delete `adapl`** (an `adapl` used before deletion of a record): if the delete statement contains a condition with which the record complies, so that the program runs through a `select no`, the record in question will not be deleted.

- **Import adapl:** if the record complies with the conditions set in the adapl, so that the program runs through a `select no`, the record will not be imported. Note that an import adapl does not import records itself: per record which is being imported, it processes data from the exchange file to allow for some data conversion before it is actually written to the target fields.
- **Print adapl:** if the adapl runs through a `select no`, printing will stop after the current record. Print adapls can either be stand-alone adapls (adapls which can be started from step 1 in the *Search wizard* or outside of Adlib) or output formats associated with a data source. Output formats are executed for every record marked for printing, while the stand-alone adapls must have built-in routines to search databases for certain records and print them; therefore their construction differs significantly. `Select no` only makes sense in output formats.
- **Link screen adapl:** if the adapl runs through a `select no`, the currently found record will not be listed in the *Linked record search screen*.
- **Select adapl:** if the adapl runs through a `select no`, the currently found record will not be included in the search result (and the reserved variable `&I` will not be incremented). This type of adapl can be called from within the *Expert search system* in an adlwin application, with the syntax: `<search statement> adapl <adapl_name>`, e.g. `all adapl findmultipleoccs`. An example of such an adapl is the following:

```
* findmultipleoccs.ada - copyright Axiell ALM Netherlands,
* 2013 - 2014
*
* Select only those records from the search result for
which
* the tag entered by the user has more than one filled
* occurrence. The user has to enter the relevant tag (tags
* are case-sensitive) when the adapl is started for the
* first time during this Adlib session. To run the adapl
* again with a different tag, restart Adlib.
* This adapl is meant to be used in the expert search
* language, in the following syntax:
* <search statement> adapl findmultipleoccs
* Of linked fields, use the link reference tag. You can't
* search on merged-in fields.
* Put the findmultipleoccs.bin file in the relevant
* application folder.
*
```

```

text tag[2]
integer occs

    /* only if the tag variable hasn't been filled during
    /* this session yet, request user input
if (tag = '') {
    locate(4,5)
    display 'Type the search tag ' + ~
    '(2 characters, case-sensitive) and press Enter:'
    locate(6,5)
    tag = input(2)
}
    /* count the number of occurrences of the tag in the
    /* current record
occs = repcnt(!tag)

    /* if tag has multiple occurrences they might still not
    /* be filled if they are part of a group
if (occs > 1) {

    /* search all occurrences above the first (assume the
    /* first is filled)
while (occs > 1) {

    /* if an occurrence is filled, end the procedure, keep
    /* this record, and continue with next record
    if (!tag[occs] <> '') {
        end
    }
    occs = occs - 1 /* count back from the highest occurrence
}
}

* if the while loop has finished normally (no filled
* occurrence above 1 has been found) or occs was 1 or 0,
* then remove this record from the search result
select no
end

```

- **Sort adap1:** if the record complies with the conditions set in the adap1, so that the program runs through a `select no`, the record will not be included in the sort. Note that sort adap1s do not actually sort records. You can use them to fill a tag with a compound value for example, and use that tag to actually sort on by entering it in the appropriate sort option in Designer or in

adlwin's sort functionality.

In the following example, a record will not arrive in the sort output if the value of the 'OM' field is equal to 0, i.e. if nothing has been filled in, if the field contains letters, or if a 0 has been entered. In all other cases, the content of the field will be formatted to 2 digits.

```
* Add leading zeros to OM field for integer sorting
* (2 characters)
if (int(val(OM)) = 0) {
  select no
  end
} else {
  OM = str$(int(val(OM)), '##')
}
end
```

Note that `select no` does not end execution of the block or loop it is in: after `select no` the next line of code is executed normally. If you do not want the rest of the adapl to be carried out, include an `end` call after the `select no` in the `if`-statement.

FACS instructions

FACS instructions provide access to database files or subfiles for searching and editing purposes. FACS (File Access Control System) uses logical file names to refer to an Adlib database or dataset. For adapls that are started from Adlib, these logical names can be defined in the application setup. It is also possible to define the logical names in the ADAPL program itself. This method can be used both for adapls that are started from Adlib and for stand-alone adapls.

The FACS system consists of the following instructions:

- **File declaration:** FDSTART, field declarations, FDEND
- **File access:** OPEN, CLOSE, CLOSEALL
- **File search:** READ
- **File maintenance:** WRITE, DELETE
- **Other:** CLEAR, LOCK, UNLOCK, ISIN, RECCOPY

ADAPL functions

ADAPL functions are built-in routines which compute a value from one

or more arguments, and then return the result to the calling program.

5.1.12 Functions

Introduction

ADAPL functions are built-in routines that compute a value from one or more arguments. You can use the result of a function for further calculations, or assign it to a variable. An ADAPL function is normally used as part of an expression. A function can also be used as an argument in another function. Each function has a unique name. A function is called in the following way:

```
variable=function(argument1,argument2,argument3, etc.)
```

The function in this example has three arguments, but in practice, the number of arguments may vary between none and many. Every function (except `MIN` and `MAX`) has a fixed number of arguments, of which the order is fixed too. For example:

```
text Name[10]
/* Ask user for name:
Name = input(10)
/* Convert first char to upper case:
Name[1] = cvt$$ (left$(Name, 1), 1)
```

Some functions are used to execute tasks, such as moving the cursor, in which case the value returned is irrelevant. ADAPL therefore allows functions to be used both in expressions and as independent instructions. For example:

```
locate(12, 35)
display 'MIDDLE'
```

Function types

Functions fall into one or more of the following categories:

- numeric functions
- operating system functions
- text functions
- record functions

- terminal functions

Numeric functions

<code>abs()</code>	Absolute value of a number
<code>asc()</code>	ASCII value of a character
<code>datdif()</code>	Number of days between two dates
<code>dayofweek()</code>	Gives the day number for the date (0 is Sunday ... 6 is Saturday)
<code>getkey()</code>	The ASCII or function code of a key
<code>instr\$()</code>	Position of a character in a string
<code>int()</code>	The whole (integer) part of a number
<code>len()</code>	The actual length of a character string
<code>max()</code>	The maximum value in a range of values
<code>min()</code>	The minimum value in a range of values
<code>mod()</code>	The integer remainder from the division of two values
<code>ndays()</code>	The number of days until (negative value) or since the beginning of 1900 (positive value)
<code>repcnt()</code>	The number of occurrences of a tag
<code>repmax()</code>	The maximum value in a tag
<code>repmin()</code>	The minimum value in a tag
<code>repsum()</code>	The total of occurrences of a tag
<code>rinstr\$()</code>	The position of a character in a string
<code>round()</code>	The rounded value of a number
<code>sqrt()</code>	The square root of a number
<code>val()</code>	The numeric value of a character string

Operating system functions

date\$()	Returns the system date as a string
getvar()	Reads the contents of a global system variable
setvar()	Puts a value in a global system variable
system()	Passes a command to the operating system
time\$()	The system time as a character string
user\$()	User name or user number as a character string

Text functions

after\$()	Returns a substring to the right of indicated character(s)
before\$()	Returns a substring to the left of indicated character(s)
chr\$()	Returns a character from the ASCII table
cvt\$\$()	Converts from/to upper case
date\$()	Returns the system date as a string
getkey()	The ASCII or function code of a key read from the keyboard
getvar()	Reads the contents of a global system variable
input()	Reads a string from the keyboard
isolatintohtml\$()	Converts a string from Isolatin to HTML
jstr\$()	Justifies (left/right) or centres a line
left\$()	The left part of a character string

<code>mid\$()</code>	Any part of a character string
<code>name\$()</code>	Changes 'surname, first name' to 'first name surname'
<code>recdate\$()</code>	Input or mutation date or time as a character string
<code>regvalidate()</code>	Validates a string to a regular expression
<code>right\$()</code>	The right-hand part of a character string
<code>round\$()</code>	The rounded value of a number
<code>setstatusbartext()</code>	Sets the text that is displayed in the status bar of the window
<code>setwindowtitle()</code>	Sets the title of the active window
<code>str\$()</code>	Converts a value to a character string
<code>string\$()</code>	Generates a string of equal characters
<code>tag2field\$()</code>	Returns the name of the tag as entered in the database setup. If no name is filled in there, the tag will be returned
<code>text\$()</code>	Reads line from text file
<code>time\$()</code>	The system time as a character string
<code>trim\$()</code>	Removes leading and/or trailing blanks
<code>user\$()</code>	User name or user number as character string
<code>whatdate()</code>	Returns a computed date as a character string

Record functions

<code>fieldisrepeated</code>	Checks if the provided field is a repeated field.
------------------------------	---

<code>fieldlength</code>	Yields the length of the provided field.
<code>fieldtype</code>	Yields the data type of the provided field.
<code>null()</code>	Deletes a field or field occurrence.
<code>onchangein()</code>	Has tag/field occurrence changed since last call?
<code>onscreen()</code>	Finds a field for the tag on the screen.
<code>reccopy()</code>	Copies a complete record.
<code>repcnt()</code>	The number of occurrences of a tag.
<code>repcopy()</code>	Copies all occurrences from tag1 to tag2.
<code>repins()</code>	Inserts an occurrence into a tag.
<code>repmx()</code>	Returns occurrence with the highest value from a tag.
<code>repmn()</code>	Returns occurrence with the lowest value from a tag.
<code>repsort()</code>	Sorts the occurrences of a tag.
<code>repsortins()</code>	Inserts data into correct occurrence of a sorted tag.
<code>repsum()</code>	The total amount of occurrences of a tag.

Terminal functions

<code>attr()</code>	Defines colours and video attributes for screen output
<code>box()</code>	Draws a box
<code>clr()</code>	Clears part of the screen
<code>getkey()</code>	Reads a character or function key from

	the keyboard
<code>input()</code>	Reads a character string from the keyboard
<code>locate()</code>	Moves the cursor
<code>onscreen()</code>	Indicates whether a certain field is on the screen
<code>show()</code>	Moves the cursor and then prints text with certain visual characteristics
<code>yesno()</code>	Asks the user to choose between two options

NB Not all functions are available in Adlib applications for DOS. These are: `milestone`, `progress`, `sendmail`, `setfont`, `setlinespacing`, `setorientation`, `setpapersize`, `setstatusbartext`, `setwindowtitle`, `wait/nowait`, `yesno`.

Obsolete functions

<code>edit\$()</code>	Edits a string of maximum length
<code>fkeys\$()</code>	Returns an escape string
<code>form()</code>	Draws a window on a terminal
<code>linkmenu\$()</code>	Brings up a link window based on tag, startvalue and endvalue, and returns the user's choice in the result
<code>omzet10\$()</code>	Converts old style (1995) Dutch telephone numbers to new 10 digit style
<code>primos()</code>	Execute an operating system command (replaced by <code>System</code> function)
<code>term\$()</code>	Returns the escape string from a specific terminal function

5.1.13 FACS

FACS (File Access Control System) is the ADAPL subsystem for searching and/or editing other* Adlib databases than the currently opened database in Adlib from which the adapl is started. (The currently opened database is the one you are working in in Adlib, so when viewing, editing or marking records, when the adapl is started.)

FACS refers to an Adlib database or dataset using logical names. For adapls that are started in Adlib, these logical names can be defined in the application setup. The logical names can also be defined in the adapl itself. This method can be used both for adapls started in Adlib and for stand-alone adapls.

The FACS instruction set:

- **File declaration:** `FDSTART`, field declarations, `FDEND`
- **File access:** `OPEN`, `CLOSE`, `CLOSEALL`
- **File search:** `READ`
- **File maintenance:** `WRITE`, `DELETE`
- **Other:** `CLEAR`, `UNLOCK`, `LOCK`, `ISIN`, `RECCOPY`

Declaring a FACS name

In an adapl you start by declaring a FACS name, with `FDSTART`. (You may have set it up partially already in the application setup.)

With the declaration (after `FDSTART`) you can also define database variables. The fields to be used (tags) are declared, with an alias for each field if required. The format of a FACS tag declaration is:

```
tag IS variable_name
```

By tag we mean the tag as it occurs in the declared file. The variable name is the name (alias) by which the field will be known and called in this ADAPL procedure. It is only necessary to assign an alias if a tag also occurs in another database in this procedure. However, it is always allowed. ADAPL programs are much clearer to read if you use aliases, which can consist of up to 32 characters, instead of tag names that can only have two. In an ADAPL program, each alias can only have one meaning.

Example 1

```
na IS name
```

ad IS address

pl IS place

Adlib distinguishes between upper case and lower case letters, so you must be consistent in your spelling of the variable names!

After defining possible database variables you close the FACS declaration with `FDEND`.

If you don't want/need to use database variables, you still have to declare the tags you want to use in this `adapl` for this FACS definition. Instead of assigning a database variable to each tag, you now just sum up the tags (one tag on each line) between `FDSTART` and `FDEND`. (And you don't need to use `IS`.)

File access

To be able to use a file with an (earlier declared) FACS name you need to open it with `OPEN`. The index and record pointers are moved to the starting position, and Adlib will initialize all database variables for this file to null. The file and accompanying variables then become available for FACS operations.

Record buffers

The database variables are stored in a so called FACS buffer; each opened FACS file has its own buffer. When you open a FACS file, this record buffer is empty because all the variables are set to null. You have to read a specific record to fill the FACS record buffer with it.

The currently opened record in your Adlib application (not an opened FACS record) during execution of the `adapl` (for instance in the case of the execution of a before-storage or after-field `adapl`), is always accessible because the current record is stored in another local record buffer called *adlib_lite*. Just use tags from this local record to access them without FACS.

If no record is currently in edit mode in your Adlib application, but you are executing an `adapl` which reads records from the currently opened database piece by piece, a print `adapl` for instance (also see Using ADAPL programs), then the currently read (local) record is stored temporarily in an implicit FACS buffer called `_LOCAL`.

So if you retrieve data from a currently opened record in a FACS database, using an `adapl`, then in the background, ADAPL uses a FACS buffer to store that record temporarily. There are three

ways of gaining access to that data via ADAPL, namely: 1. by including the tags from that record in a FACS declaration and addressing the aliases, 2. via tag indirection, or 3. (available from 6.5.2) by using the record buffer name (the FACS name or *adlib_lite*), followed by an exclamation mark and the tag you wish to address. Here are four examples of the third method (assume *Catalogue* and *People* are declared FACS databases):

```
display Catalogue!ti ...
If (len(Catalogue!ti) = 0) { ...
errorm adlib_lite!do ...
if (enumval$(People!do, People!do, -1) = 'AUTHOR'){ ...
```

You can change the values in FACS-declared tags or aliases all you want: those changes won't ever be stored in the relevant record automatically. Only an explicit `WRITE` of the relevant FACS database will store any changes made to the buffer in the record opened by FACS.

The situation is different for the record currently opened in your Adlib application during execution of the `adapl`. Although its contents are stored in the *adlib_lite* record buffer, any changes you make to values in tags will appear in the opened record immediately, without having to explicitly write the record. The new values will of course only be actually stored, if the user saves the record (or was already saving the record) and the new field contents pass validation and such.

However, when you want to write changes to records in the currently opened database in your application (not an opened FACS file) from within a `print adapl` or other `adapl` which is not executed while a record is in edit mode, you do have to write these changes explicitly. (This is because if no record is in edit mode, the *adlib_lite* buffer will be empty.) In this case you'll have to treat the local buffer as a FACS record. You address it with the non-declared "FACS" name `_LOCAL`. Use a `WRITE` to `_LOCAL` to write the changes to the currently read database record.

File search

To search through a file declared with FACS you use the `READ` instruction. You can read files as well as pointer files.

File maintenance

During a read, a found record is being 'locked' to prevent others on

your system to delete or edit this record while your adapl wants to perform some action on this record. After a `WRITE` or `DELETE` on the current record the lock on it is automatically removed. When no action is performed on a found record you must unlock the record with `UNLOCK`.

After a system crash or when using adapls that do not correctly unlock records after locking them, you might find it impossible to edit certain records when working in an Adlib application. Errors 34 or 85 or messages appear telling you that a record is already in use or that no record lock can be applied. These record locks must then first be removed before you can edit or delete the record. Preferably, use the *Record lock manager* tool for this purpose. A more rudimentary way of removing locks is by removing the `.lck` file with the name of the appropriate dataset in the `data` directory of your Adlib system (applies only to CBF databases); but before doing so, make absolutely sure no-one is currently working with Adlib!

Use `WRITEEMPTY` to create an empty record in the file referred to by `facname`. With this you can obtain the record number of this record before you store it.

With `DELETE` you remove the current record from the file.

Other FACS instructions

Function	Short description
<code>UNLOCK</code>	Removes the write lock to allow others to edit it again.
<code>LOCK</code>	Locks a record to edit or delete it.
<code>CLEAR</code>	Clears all database variables declared for the corresponding <code>facname</code> from the record.
<code>ISIN</code>	Checks whether a <code>preref</code> is in a dataset.
<code>RECCOPY</code>	Copies a FACS record to a different FACS definition.

File closing

After your work on FACS files in an adapl is done, you need to close the files you opened with `OPEN`: use `CLOSE` or `CLOSEALL`. The result code of the operation is put in the system variable `&E`. The file and corresponding variables are then no longer available. If any record

locks remain, Adlib will display a warning message and removes the locks.

Note that FACS files are closed automatically as well, after processing of the adapl is done. To not use `CLOSE` explicitly may speed up processing because opening a file again and again for each record in a selection of records to be processed takes time.

FACS error handling

The reserved variable `&E` is given a status code after each FACS instruction. If an instruction succeeds, then `&E=0`; any other values indicate that the operation was not successful. Typical reasons for the failure of a FACS operation include: no records were found using the given search key (`&E=7`), and, a record was found but could not be locked (`&E=34`). With `READ`, `&E=55` means that the file could not be opened, and with `OPEN`, that the file or FACS name was incorrect.

You can look up the exact meaning of any value of `&E` after a FACS instruction in the list of error codes. Often, the exact reason for failure of an operation is not relevant, and it suffices to check whether `&E` equals zero.

The variable `&E` can also be written to by the ADAPL program. It is therefore important to check `&E` immediately after every FACS instruction.

When the program ends, the last value of `&E` is returned to the calling program as exit code. Unless you assign a different value to `&E` in the ADAPL program, the result of the last FACS instruction is passed to the calling program.

5.1.14 Handling text files

With ADAPL, you can use texts that are read from separate text files. The text should meet the following conditions:

- The text must be in ASCII format.
- Each line must begin with an ascending line number, starting with 1, without skipping any numbers. If you want to reserve numbers for future use, you should give the line a number, but leave the rest of the line empty.

This construction makes it possible to distinguish between the number and the first character of the text line, which may also be a number. Otherwise it would be impossible to tell what was the number of the text line and what was the text itself.

- Comment lines start with an asterisk (*).
- There are no spaces between line number and text. If there is a blank space, it will be considered as part of the text and will therefore be printed.

Example of a text file:

```
* date.txt
1Error: you can only enter today as the date.
2Error: the week number you entered does not exist.
```

The following functions are available for use with text files:

Instruction	Brief description
openfile	Opens text file (to read texts from it)
closefile	Closes text file (from which texts are read)
closeallfiles	Closes all files opened with openfile
text\$	Fills text variable with text from a certain line number

It is also possible to put all texts used by adapls in one text file (per language), and open the right language variant in the running application automatically, whenever an adapl needs to read from a text file, by just providing the name of this *.txt* file once, in the application setup on the *Advanced* properties tab of the application. The advantage of this option is that you no longer have to open specific text files in adapls before you can read from them.

An example in ADAPL:

Let us take the text file named *date.txt*, and the adapl *date.ada*. This is a screen adapl that is carried out after a new record has been entered. The adapl checks whether today's date has been filled in as date (tag: DA) and whether a valid week number has been filled in (tag: WE):

```
*date.ada
*Checks date and week field.
integer texthandle
* open textfile:
texthandle = openfile('date.txt',0)
if (texthandle=-1) { /*error
errorm 'Error opening textfile'
```

```

end
}
if (&l=3) { /*after input
if (DA <> date$(7)) {
errorm text$(texthandle,1) /*invalid date
errorf DA
redisplay
}
if (val(WE)=0 or val(WE)>53)) {
errorm text$(texthandle,2)
errorf WE
redisplay
}
}
closefile (texthandle) /*close textfile
end /*end of adap1

```

Multilingual

If you want your application to be available in a number of different languages, you will not be able to have fixed texts in the adapls. In other words, you will have to use text files for the various languages. For example the following situation: an adap1 is executed in the *serials* subdirectory. The active language is language number 2: French. The adap1 will contain:

```
handle = openfile('serials.txt', 0)
```

Adlib will look for the appropriate text file as follows:

1. Adlib will 'translate' *serials.txt* into *serials2.txt* and will search for this file in the current directory (*serials*). If the file is not found there:
2. Adlib will search for the text file *serials.txt* in the subdirectory to which the `ADLIB_DIR2` environment variable refers, e.g. `\Adlib\FRENCH`. If the file is not found there:
3. Adlib will search for the file *serials.txt* (in the default language) in the current directory. If the file is not found there:
4. Adlib will search for *serials.txt* in the `ADLIB_DIR` directory. If the file is not there either, then:
5. An error message will be displayed.

To find out which language is the active language, you can consult the reserved tag &P. Its value is the number of the active language. For example:

&P=0 Default language

&P=1 Language 1

etc.

5.1.15 Using Word templates

The ADAPL functions `WordCreateDocument`, `WordCreatelabels` and `WordPrintDocument` can be used to fill a Word template with Adlib data, from an adapl.

If you are using this adapl in a stand-alone situation, the adapl does not know in which dataset or database it should look for data, and you will have to modify a template for this use. Instead of a single field reference, you now use the FACS name of the desired dataset or database, followed by a plus character and the field tag (or field name, but that is not recommended), so: `<<DesiredFACSDatabase+DesiredTag>>`

If you have set an adapl as an output format, the adapl can find the local database (the database in which the currently selected records are located) and there is no need to include the FACS name in the field reference. But any different database that you have opened in the adapl by means of FACS, can only be accessed in a template through the full syntax: `<<DesiredFACSDatabase+DesiredTag>>`

See also

Output formats in the application setup

ADAPL programming: output formats

`WordCreateDocument`

`WordCreateLabels`

`WordPrintDocument`

FACS

5.1.16 Output formats

Output formats are special ADAPL programs that are used to print or save marked records. When output is sent to the printer, it is always sent via the default printer port. In a network, the output usually goes to the network printer. The ADAPL program is used to format the printout of the records. To use output formats, take the following steps:

1. Write the ADAPL program using any text editor.
2. Compile the adapl with the ADAPL compiler. This produces an executable program file with the extension *.bin*.
3. In the *Output jobs* list under a data source node in the application setup, you add the print adapls to be used for the currently selected data source.

Please note that, in the case of multilingual applications, there are special conditions that have to be met for the title, and fixed texts in the adapl itself are not possible; instead, text files have to be used.

Example

Text of the ADAPL print format contained in the file *LIJST.ADA*:

```
title 'LIST PRIREF AND 1 TITLE LINE'
*-----
print 3, 8, str$(%0, 'ZZZZZZ#') /* priref
print 15, 60, ti[1] /* title occurrence 1
output line /* with line break
end
```

See also

Output formats in the application setup

ADAPL programming: accessing data for a Word template from within an adapl

5.1.17 F12 adapl

F12 adapls can only be used with Adlib for DOS (under Windows, the same functionality is achieved in a different way). These adapls are executed when the user presses **F12**. There are three types of F12 adapls: one executed in display mode, one executed in edit mode, and one executed from a link screen.

- Display mode: the adapl named *f12adapl.bin*.
- Edit mode: the adapl named *linkproc.bin*.
- Link screen: the adapl you enter under command procedure.

The F12 ADAPLs can be used for all kinds of purposes. The most common application has proved to be starting up WordPerfect in Adlib, so that a WP document can be viewed in Adlib. An example:

```
text command[80]
```

```

if (bn <> '') {
  /* in bn a unique file name is saved
  /* e.g. doc1.wpd
  command = 'dowp c:\wp51\' + bn
  /* dowp is a batchfile that starts wp and then
  /* opens the file in bn that is located in C:\wp51\
  system (command)
  attr(blue on white)
  cls
}
end

```

The corresponding batchfile *dowp.bat* looks like this:

```

@echo off
echo 'Please wait.'
MODE CO80
if exists %1 C:\wp51\wp %1

```

This is but one example. The possibilities are legion.

5.1.18 Running an adapl

Normally adapls are automatically run after some user action in an Adlib application when you are already working in a database, like when leaving a field or a screen, when opening or saving a record or after starting a print or export job. However, it is also possible to run a stand-alone ADAPL program as an option from step 1 in the *Search wizard* of a running application, although you would then have to structure the adapl differently, because normally the entire adapl is executed for the currently loaded database record automatically. The same applies to ADAPL procedures run outside an Adlib application, which is possible as well. In the latter case, to start a compiled adapl from a (DOS) command-line window, you have to use *adeval.exe* from your Adlib *\bin*, *\executables* or *\tools* directory. Run a stand-alone adapl from the command-line as follows:

```
adeval adapl_name [arguments]
```

For the adapl name, fill in the name of an ADAPL executable file. You can leave out the extension *.bin*. The arguments are an optional list of values that the adapl must use. The arguments must be separated by blank spaces. Arguments are only required if expected by the ADAPL program.

When an ADAPL program ends, it passes control back to the caller (i.e. the operating system or Adlib). It may also pass back a code. For example, under DOS, *adeval.exe* will pass the exit code of the adapl

back as 'error level', so that a script can use the result of the ADAPL procedure to determine what should happen next.

Note that the current Windows version of `adeval.exe` and the built-in version in `adlwin`, support 32-bit binary `adapl` files as well as the old 16-bit code. (The runtime engine automatically detects which version of the compiler has been used.) The `DOS4GW.exe` program is no longer required to run this version of `adeval.exe`.

Eval

Both `adeval.exe` and the built-in version in `adlwin.exe` use `eval`: `Eval` is a function library for interpreting binary (compiled) ADAPL programs. `Adeval.exe` really is `eval` with a command-line interface, while in `adlwin.exe` it is hidden from the user. `Eval` itself was originally written as so-called unmanaged code in the C and C++ programming languages, just like `adlwin.exe` and `wwwopac.exe`. Currently (2013), old code in different Adlib software is gradually being rewritten as managed code in the C# programming language, which allows for safer code and faster development.

Managed code is safer than unmanaged code because in managed code array boundaries are checked, while in unmanaged code pointers are used. Also, managed code does not compile directly to machine code, but to intermediate code for which interpretation the Microsoft .NET Framework is required on the computer that is going to run the managed code.

More recent Adlib products (like `wwwopac.ashx`, the Museum Tracker pda software, and the workflow software) are being written in C# completely, and to allow these products to execute `adapls`, `eval` needs to be rewritten in C# as well because the managed .NET environment cannot execute the unmanaged `eval` code. The new `eval` machine is currently a work in progress; still, a core set of functionality has already been implemented. Even though as of yet the new version is not available to users, the Designer Help documentation of ADAPL functions will sometimes already refer to features only available in the C# version of `eval`.

The core set of functionality currently (April 2013) implemented in the C# version of `eval`, is the following: `ABS`, `AFTER$`, `ASC`, `BEFORE$`, `CHR$`, `CLEAR`, `CLOSE`, `CVT$$`, `DATDIF`, `DATE$`, `DAYOFWEEK`, `DISPLAY`, `ENUMVAL$`, `ERRORF`, `ERRORM`, `GETVAR`, `GUID`, `INSTR$`, `INT`, `JSTR$`, `LEFT$`, `LEN`, `LOCK`, `MAX`, `MID$`, `MIN`, `MOD`, `NDAYS`, `NULL`, `OPEN`, `OPENFILE`, `OUTPUT`, `PDEST`, `PRINT`, `READ`, `REDISPLAY`, `REPCNT`, `REFIND`, `REPINS`, `REPMAX`, `REPMIN`, `REPSORT`, `REPSUM`, `RIGHT$`, `RINSTR$`, `ROUND`, `ROUND$`, `SENDMAIL`, `SETVAR`, `STRING$`, `TEXT$`, `TIME$`, `TRIM$`, `UNLOCK`, `USER$`, `VAL`, `WHATDATE`, `WRITE`, `WRITEEMPTY`, `YESNO`.

Available reserved variables are `&0`, `&1`, `&4`, `&6`, `&E`, `&F`, `&H` and `&P`.

The list below contains a list of reserved words in the C# version of eval and is complementary to the list of core functions. The list comprises functions and instructions, internal variable or constant names, and words reserved for future use: and, do, else, end, gosub, goto, if, include, let, not, numeric, or, return, select, status, then, title, until, while.

5.2 Reference

§

5.2.1 ABS

Syntax

```
num2 = ABS(num1)
```

Arguments

```
num1, num2: numeric
```

Meaning

The `abs` function yields the absolute value of `num1` (the difference between `num1` and 0, so always a positive number).

Example

```
Integer value  
value=abs(-10)
```

Result

```
value is 10
```

5.2.2 AFTER\$

Syntax

```
result = AFTER$(begin, text, search_key)
```

Arguments

```
text, search_key, result: text
```

```
begin: integer
```

Meaning

After\$ searches in string `text` from position `begin` for character or substring `search_key`, and puts all characters to the right of that character or that substring in `result`. If `search_key` is not found in `text`, `result` will remain empty.

Example

```
text1 = after$(3, 'z,zzzz,yyyyy', ',')
```

Result

```
text1 is 'yyyyy'.
```

See also

BEFORE\$

5.2.3 ASC

Syntax

```
int = ASC(text)
```

Arguments

```
int: integer
```

```
text: text
```

Meaning

Gives the ASCII value of the first character of `text`. An empty `text` string returns the value 0.

In the new C# implementation of `eval`, the Unicode value of the first character of `text` is returned; this means that in this case `text` may also contain (Unicode) characters not available in ASCII.

Example 1

```
display ASC('B')
```

Result

With most computer systems, the result will be 66. The `ASC` function uses the ASCII character table defined for the host computer. With a different system this function may return a different value.

Example 2

```
display ASC('')
```

Result

0

Example 3 (Unicode)

```
display ASC('Ω and other special characters')
```

Result

937

See also

`CHR$`

5.2.4 ATTR

Syntax

```
ATTR(video foreground ON background)
```

Arguments

See table below.

Platform

DOS

Meaning

Set the colours and video attributes for the screen. Possible properties:

video	foreground	background
BLINKING	BLUE	BLUE
BRIGHT	YELLOW	RED
DIM	RED	GREEN
REVERSE	GREEN	CYAN
	CYAN	MAGENTA
	MAGENTA	WHITE
	WHITE	

Example

```

INTEGER I
I = BRIGHT BLUE ON GREEN
ATTR(I)
DISPLAY 'This text is bright blue on green'

```

Result

The text *This text is bright blue on green* will appear in bright blue against a green background.

5.2.5 ATTRIB

Syntax

```
attrib [+R|-R] [+H|-H] [<drive>:][<path>]<file name> [/D]
```

Meaning

Assign an attribute to one or more files. You can use wildcards (*). This function can only be used in ADAPL as the parameter in a SYSTEM command.

+ sets an attribute.

- removes an attribute.

R is the *Read-only* attribute.

H is the *Hidden* attribute.

/D also processes directories.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Example

```
stat = SYSTEM('attrib +R overdue.txt')
```

5.2.6 BEFORE\$

Syntax

```
result = BEFORE$(begin, text, search_key)
```

Arguments

text, search_key, result: **text**

begin: **integer**

Meaning

Search in string `text` from position `begin` for character or substring `search_key` and put all characters to the left of that character or substring in `result`.

If `search_key` is not found in `text`, the entire `text` is placed in `result`.

Example

```
string = before$(3, 'z,zzzz,yyyyy' , ',')
```

Result

string is 'z,zzzz'.

See also

AFTER\$

5.2.7 BINPATH

Syntax

```
BINPATH '<file name>.bin'
```

Meaning

This optional command instructs *adapl.exe* on compilation of the *.ada* file to a *.bin* file, to compile it to the provided path and/or file name only. If only a path is provided, the *.bin* file will get the name of the source file.

The instruction may appear anywhere in the *.ada* file, and may be repeated to compile the file to different locations/names at once.

Example

Suppose you insert the following instructions in a file called *orderlist.ada* in the Adlib *\adapl sources* folder:

```
binpath 'somefile.bin'  
binpath '..\test\orders.bin'  
binpath 'C:\adlib software\adaplb\bin'
```

Result

On compilation, three *.bin* files will be created: *somefile.bin* in the same folder as the *.ada* source file, *orders.bin* in the *\test* folder (at the same level as *\adapl sources*) and *orderlist.bin* in *C:\adlib software\adaplb\bin*

5.2.8 BOX

Syntax

```
stat = BOX(row1, col1, row2, col2)
```

Arguments

```
row1, col1, row2, col2, stat: integer
```

Platform

DOS

Meaning

Draws a rectangle with as coordinates for the top left corner: row `row1`, column `col1`, and as bottom right corner, row `row2`, column `col2`. The result is '0' if the operation is successful, otherwise '1'.

Example

```
BOX(10, 10, 15, 15)
```

Result

Draws a rectangle from row 10, column 10 to row 15, column 15.

5.2.9 BREAK

Syntax

```
BREAK ON
```

```
BREAK OFF
```

Meaning

Determines whether the user can or cannot break the ADAPL program (`ON` or `OFF`, respectively) with `ctrl-c` or another key used for interrupting programs. When starting up an ADAPL program, Adlib assumes the value `OFF`.

5.2.10 CHDIR

Syntax

```
chdir [/D] [<drive>:]<path>
```

or

```
chdir [..]
```

or

```
cd [/D] [<drive>:]<path>
```

or

```
cd [..]
```

Meaning

Make the indicated directory active, or display the name of the currently active directory. This function can only be used in ADAPL as the parameter in a SYSTEM command.

.. indicates the directory above the currently active one.

Type `cd <drive>`: to display the active directory on the relevant drive. Typ just `cd` without parameters, to display the active drive and directory.

Use `/D` if you want to change both the active drive and the active directory.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Examples

```
stat = SYSTEM('chdir /D "C:\my docs\''')
stat = SYSTEM('chdir /D C:\printed_docs\')
```

5.2.11 CHR\$

Syntax

```
result = CHR$(number)
```

Arguments

result: text

number: integer

Meaning

CHR\$ returns the ASCII character for the ASCII value `number`, a number between 0 and 255.

On Unicode Windows systems, and only in the new C# implementation of eval, a value between 0 and 65536 can be used.

Example 1

```
display CHR$(66)
```

Result

With most computer systems, a *B* will be printed. The function `CHR$` uses the ASCII table of the host computer system. On a different system, this function may return a different character.

Example 2

```
textvar = "This is line 1" + CHR$(13) + CHR$(10) + ~
         "This is line 2"
```

Result

With most computer systems, `CHR$(13) + CHR$(10)` means a carriage return plus line feed. So the two texts will be printed underneath each other when `textvar` is printed. Use this syntax if you want to include the new line in a text you are putting together. If you are sending text to the printer directly, it's easier to use the output line command.

Example 3 (Unicode)

```
display CHR$(937)
```

Result

Ω

Comments

`CHR$` can be used to send escape codes to the printer.

Syntax

```
PDEST printer port
PRINT pos, max, chr$(27) + 'code'
OUTPUT
```

Example 4

```
pdest 'lpt1'
print 1, 80, chr$(27) + 'string_escapecodes'
output
```

See also

ASC

Your printer manual for the escape codes

5.2.12 CLEAR

Syntax

```
CLEAR facsname
```

Meaning

Clears all database variables declared for facsname, except for the primary record number. This does not mean that the record in question has already been 'deleted' from the database. That doesn't happen until you write it with `WRITE`.

See also

[Click here for more information about how to define a FACS name in the application setup.](#)

[Click here for general information about FACS.](#)

5.2.13 CLOSE

Syntax

```
CLOSE facsname
```

Meaning

Closes the file referred to by facsname and resets the error status. The file and corresponding variables are then no longer available for FACS operations.

If any record locks remain, a warning message is displayed and the locks are removed.

See also

[Click here for more information about how to define a FACS name in the application setup.](#)

[Click here for general information about FACS.](#)

5.2.14 CLOSEALL

Syntax

```
CLOSEALL
```

Meaning

Closes all files opened with `OPEN`.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.15 CLOSEALLFILES

Syntax

```
CLOSEALLFILES
```

Meaning

Closes all text files that have been opened in the ADAPL (so all files opened with `OPENFILE`).

See also

Handling text files and `OPENFILE`

5.2.16 CLOSEFILE

Syntax

```
CLOSEFILE (handle)
```

Arguments

```
handle: integer
```

Meaning

Closes the text file referred to with `handle`.

See also

Handling text files and `OPENFILE`

5.2.17 CLR

Syntax

```
stat = CLR(row1, column1, row2, column2)
```

Arguments

```
stat, row1, column1, row2, column2: integer
```

Platform

DOS

Meaning

Deletes a rectangular part of the screen with as coordinates row `row1`, column `column1` as top left corner and row `row2`, column `column2` as bottom right corner. The cleared areas of the screen assume the background colour. The result is '0' if the operation is successful, and '1' if unsuccessful.

Example

```
CLR(10, 10, 15, 15)
```

Result

Clears a rectangle from row 10, column 10 to row 15, column 15.

5.2.18 CLS

Syntax

```
CLS
```

Platform

DOS

Meaning

Deletes all text from the screen. The whole screen is filled with the background colour.

5.2.19 COLUMN

Syntax

```
COLUMN (pos1, pos2, pos3, ...)
```

Arguments

pos1, pos2, pos3: integer

Meaning

Determines that Adlib is to print in columns. The columns start at the indicated positions (pos1, pos2, pos3, ...). The number of columns therefore depends on the number of start positions that you enter.

Dimensions for this function are in AUs (Adlib Units). A horizontal AU is 1/90 of the width of a standard printable A4 area; a vertical AU is 1/72 of the length of the standard printable A4 area. (The standard font and font size for printing exactly 90 characters across the full available page width, is Courier size 11. If you change the margins yourself, then correspondingly the number of units that can be printed will be reduced. If you change the font you will be able to print fewer or more characters across the width of the page, depending on the font and font size you choose.)

Example

```
COLUMN(1, 41)
```

Result

Printing will be in two columns, with column 1 starting at position 1 and column 2 starting at position 41.

5.2.20 COPY

Syntax

```
copy [-Y] [/L] [/A|/B] <source> [<target>]
```

Meaning

Copy one or more source files in your file system to a different (target) location (a folder and/or file names for the new files). You can use wildcards (*). This function can only be used in ADAPL as the parameter in a SYSTEM command.

/A indicates that it concerns an ASCII file.

/B indicates that it concerns a binary file.

/-Y asks for confirmation before overwriting a target file.

/L copies the source link towards the target instead of the file to which the source link points, if the source is a symbolic link.

By default you are warned before a file is overwritten, unless `copy` is executed from within a batch script.

To concatenate files into one, provide one file as the target and multiple files as the source: use wildcards or the notation: `file1+file2+file3...`

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Example

```
stat = SYSTEM('copy /-Y /A *.txt C:\myfiles\')
```

5.2.21 COPYFILE

Syntax

```
result = COPYFILE ('source', 'destination')
```

Arguments

result: integer

source, destination: text

Meaning

Copyfile copies the file in source to destination. Specify the whole paths. The result is an Adlib error code.

5.2.22 CURSOR

Syntax

```
CURSOR tag[occ]
```

Meaning

Adlib will place the cursor in the specified occurrence of the field (under Windows and DOS); an occurrence number is mandatory. For tag, you fill in the (tag)name of the database variable. For occ (occurrence), you enter the number of the iteration.

5.2.23 CVT\$\$

Syntax

```
result = CVT$$ (text, code)
```

Arguments

text, result: text

code: see table below

Meaning

Converts characters in text according to the key value code.

Code	Meaning
1	Text is converted to upper case.
2	Text is converted to lower case.
4	Accents are removed from the characters in text.

5	Text is converted to upper case and accents are removed.
6	Text is converted to lower case and accents are removed.
10	Text is converted to lower case, starting with a capital.
32	Text is converted from DOS to ISO-Latin. Do not use this option in output adapls with which you print to file, from Adlib 6.0 or higher, since it may cause diacritical characters to be displayed incorrectly. This is because Adlib Unified internally works with Unicode in UTF-8 encoding (even though your databases may still be stored in the DOS/OEM or ANSI/IsoLatin character set).

Example

```
name = CVT$$('Smith', 1)
```

Result

The variable `name` is filled with `SMITH`.

5.2.24 DATDIF

Syntax

```
days = DATDIF(date1, date2)
```

Arguments

`days`: integer

`date1, date2`: text

Meaning

Parameters `date1` and `date2` must be valid dates in the Julian, European or ISO format. The American format is not supported (see the description of the `DATE$`-function). This function gives the number of days between the dates `date1` and `date2`. If `date2` is later than `date1`, the result will be positive; if `date2` is earlier than `date1`, the

result will be negative. For both dates, you can also use 'today' or date\$(<code>). The system will then automatically fill in the value that is valid when the adapt is carried out. It is also possible to combine 'today' or date\$(<code>) with a plus or a minus sign followed by an integer (e.g. 'today' -10). That signifies the system date plus or minus the stated number of days.

Example

```
difference = datdif('22/11/62', '07/08/62')
```

Result

The variable `difference` is filled with `-107`

See also

DATE\$

5.2.25 DATE\$

Syntax

```
date = DATE$(code)
```

Arguments

date: text

code: see the table below

Meaning

Returns the system `date` as a character string. The `code` determines how the date is presented.

Code	Meaning
1	in the form <i>mm/dd/yy</i> (USA format)
2	in the form <i>yy.ddd</i> (Julian format)
3	in the form <i>dd/mm/yy</i> (European format)
4	in the form <i>yy-mm-dd</i> (ISO format)

code + 4	as above, but with a 4-digit year
----------	-----------------------------------

Example

```
date = date$(8)
```

Result

The variable `date` is filled with the current system date, at the time of executing this line of code, in this case: *2002-05-13*

5.2.26 DAYOFWEEK

Syntax

```
day_number = DAYOFWEEK(date)
```

Arguments

`day_number`: integer

`date`: text

Meaning

Parameter `date` must be a valid date in the Julian, European or ISO format. The American format is not supported. This function returns the number of the day of the date entered, where *0* stands for Sunday, *1* for Monday, ... and *6* for Saturday. For `date` you can also use `'today'` or `date$(<code>)`. The system will then automatically fill in the value that is valid when the ADAPL is executed. It is also possible to combine `'today'` or `date$(<code>)` with a plus or a minus sign followed by an integer (e.g. `'today' - 10`). This represents the system date plus or minus the number of days stated.

Example

```
day_number = dayofweek('12/09/1995')
```

Result

Returns the value *2* for `day_number`.

See also

DATE\$

5.2.27 DEL

Syntax

```
del [/P] [/F] [/S] [/Q] <file names>
```

or

```
erase [/P] [/F] [/S] [/Q] <file names>
```

Meaning

Delete one or more files from your file system. You can use wildcards (*) to remove multiple files at once. If you enter a directory name, all files in it will be deleted. This function can only be used in ADAPL as the parameter in a SYSTEM command.

/P asks for confirmation of every deletion.

/F forces removal of read-only files.

/S removes the indicated files from all subdirectories as well.

/Q in combination with wildcards, removes files without asking for confirmation.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Example

```
stat = SYSTEM('del /P /S *.txt')
```

5.2.28 DELETE

Syntax

```
DELETE facsname
```

Meaning

The DELETE instruction removes the current record read in with UPDATE from the file. If the action is successful, the status variable &E will be

set to 0. If unsuccessful, &E will be unequal to 0.

It's important to lock a record prior to each type of `WRITE` operation, `WRITEEMPTY` or `DELETE`, and unlock it afterwards via `LOCK facsname` and `UNLOCK facsname` respectively. This prevents a record being kept locked unintentionally after editing, which would make it impossible for others to edit it again.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.29 DELETEFILE

Syntax

```
result = DELETEFILE ('file_name')
```

Arguments

result: integer

file_name: text

Meaning

Deletfile deletes the specified file. The `result` is an Adlib error code.

5.2.30 DISPLAY

Syntax

```
DISPLAY text_expression
```

Platform

DOS

Meaning

Displays the `text_expression` on the screen in a box, starting at the cursor position. (The cursor position can be specified using `locate(line, column)`.)

Example

```
locate (6, 5)
display '*** Hello Adlib user ***'
```

5.2.31 DO UNTIL

Syntax

```
DO instruction UNTIL expression
OR:
DO {
instruction
instruction
...
} UNTIL expression
```

Meaning

The `instruction` or the `instruction block` is carried out at least once. If the `expression` is `FALSE`, it is repeated until the `expression` is `TRUE`.

See also

[Click here for more information on conditional loops.](#)

5.2.32 END

Syntax

```
END
```

Meaning

Execution is stopped immediately. The error status `&E` is returned as exit code to the calling program (an Adlib application, the operating system or a batch program).

See also

[Click here for more information on control instructions.](#)

5.2.33 ENUMCODE\$

Syntax

```
neutralvalue = enumcode$ (tag[occ])
```

Arguments

neutralvalue: text

tag: database variable

occ: integer

Meaning

This function is a simplified version of `enumval$`: it retrieves the neutral value of the displayed value in the specified enumerative field occurrence (the field must be defined in the database setup as enumerative).

Note that an enumerative field allows the user to choose a value from a fixed drop-down list. This value is presented in the current interface language. However, this is not the value which is stored in the record: that would be the neutral or standard value. Neutral values and their "translations" for display-only are specified with the enumerative field in the data dictionary.

Example

Assuming the values of enumerative field T1 are:

```
#0000CC (Neutral value)  
Blue (Language number 0: English)  
Blauw (Language number 1: Dutch)
```

```
#FF0000 (Neutral value)  
Red (Language number 0: English)  
Rood (Language number 1: Dutch)
```

```
#33CC00 (Neutral value)  
Green (Language number 0: English)  
Groen (Language number 1: Dutch)
```

With the interface language in `adlwin.exe` set to English, an example record has two occurrences of T1, the first displaying the value *Red*, the second *Blue*. Via ADAPL we could retrieve the neutral value of both occurrences as follows (if `neutralvalue1` and `neutralvalue2` have been declared as text variables):

```
neutralvalue1 = enumcode$(T1[1])
neutralvalue2 = enumcode$(T1[2])
```

Result

```
neutralvalue1 will be: #FF0000
neutralvalue2 will be: #0000CC
```

5.2.34 ENUMVAL\$

Syntax

```
translation = enumval$ (source_text, tag, language_number)
```

Arguments

```
translation, source_text: text
```

```
tag: database variable
```

```
language_number: integer
```

Meaning

This function "translates" a `source_text` which must appear in the tag `tag` (the field must be defined in the database setup as enumerative) into the corresponding text in the language with the number `language number`. This can only be done if for the requested language the translation is already present in the tag `tag`. The result is placed in the variable `translation`. The translation will be an empty string if no corresponding value is found.

In a record, in this field only the language number is stored.

The standard text for the field in this record can be read using language number `-1`.

Example

Assuming the texts of field T1 are:

```
#FF0000 (Language-independent value)
Red (Language number 0: English)
```

Rood (Language number 1: Dutch)
Rouge (Language number 2: French)
Rot (Language number 3: German)

```
translation1 = enumval$('Purple', T1, -1)
translation2 = enumval$('Rot', T1, 0)
translation3 = enumval$('Rot', T1, 3)
translation4 = enumval$('Rot', T1, -1)
translation5 = enumval$('Rot', T1, 4)
translation6 = enumval$('Rot', T2, -1)
```

Result

translation1 will be: "", because *Purple* does not occur in T1.
translation2 will be: *Red*.
translation3 will be: *Rot*.
translation4 will be: *#FF0000*.
translation5 will be: "", because no value is filled in for language number 4 in T1.
translation6 will be: "", because *Red* does not occur in T2.

5.2.35 ERRORF

Syntax

```
ERRORF tag[occ]
```

Meaning

The entry field for `tag[occ]` will start blinking (under DOS) at the next `REDISPLAY`. `Adlib` will place the cursor in the specified occurrence of the field (under Windows and DOS); an occurrence number is mandatory. Under Windows, a `REDISPLAY` is not required to put the cursor in the specified field occurrence. For `tag`, you fill in the (tag)name of the database variable. For `occ` (occurrence), you enter the number of the iteration. If `[occ]` is not included, all occurrences of the field will blink (under DOS).

Note that the field `tag[occ]` must be present on the current detailed display screen tab when the `adapl` is being executed. `Adlib` will not switch tabs if the field is on another tab than the active one.

5.2.36 ERRORM

Syntax

```
ERRORM text_expression
```

Meaning

Displays the `text_expression` on the Adlib screen. The user is asked to press **Enter** (or click *OK*, under Windows) to continue. The text will then disappear from the screen.

5.2.37 ERROR\$

Syntax

```
result = ERROR$ (errorcode)
```

Arguments

result: text

errorcode: numeric

Meaning

The function `error$(errorcode)` provides extra information about the last generated error code. This information consists of data generated by the software, such as the name of the database that couldn't be opened or the name of an index file that appears corrupted. (So no text file is used.) The last generated error code is kept in the reserved variable `&E`.

Example

In adapls you can display the result of `error$(&E)` in an error message for the user or system administrator, next to the error code itself:

```
open document
if (&E) {
  errorm 'Error ' + &E + 'Details: ' + error$ (&E)
  redisplay
} else {
  errorm 'No error '
  redisplay
}
```

Result

If during opening of the dataset no error occurs, then the message *No error* will be shown. If an error does occur, then the error code is displayed, preceded by the word *Error* and followed by *Details*: and the extra error information.

5.2.38 FDEND

Syntax

```
FDEND
```

Meaning

FDEND finishes a FACS file definition. The facsname is a logical name that refers to an Adlib database. The actual name and location in the file system of the host computer can be specified with Adlib Designer.

Example

Example of a FACS file declaration:

```
FDSTART BOOK
au IS BOOK_Author
ti IS BOOK_Title
%0 IS BOOK_Priref
FDEND
```

Result

BOOK_Author then refers to the field *au* in the database *BOOK*, etc.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.39 FDSTART

Syntax

```
FDSTART facsname '<path>+<database file name>'
```

Meaning

`FDSTART` starts a FACS file definition. The `facsname` we define here, is a logical name (alias) that refers to an Adlib database or dataset. Besides the `facsname`, you need to provide the (relative) path plus the file name of the database to enable the ADAPL compiler to make the link to the actual database. When you use a relative path, make sure it is relative to your Adlib application folder(s) containing the `.pbk` file(s). The file name is given as a character string between single quotation marks, starting with the location (directory) of the database file in the file system, then a 'plus' sign, then the name of the database in the file system (without extension), possibly followed by a 'greater than' sign and the dataset name.

Example 1

Example of a FACS file declaration if you don't want to assign database variables because you're only using one database in this `adapl`:

```
FDSTART DOCS '../data+document'  
  
au  
  
ti  
  
%0  
  
FDEND
```

Result

Each declared tag here, from the FACS database `DOCS`, can now be accessed in this `adapl` directly. (If you use tags in the `adapl` that are not declared in an `FDSTART ... FDEND` block then those tags must reside in the local record buffer instead of the FACS buffer that you declare. This buffer is implicitly declared and has the name `adlib_lite`. The buffer contains the currently opened/processed record by `adlwin.exe`, and is therefore not available for stand-alone `adapls`.)

Example 2

To declare the `book` dataset in the `document` database in the directory `../data` under the alias `BOOKS` with the definition of three

database variables, code as follows:

```
FDSTART BOOKS '../data+document>book'  
au IS BOOK_Author  
ti IS BOOK_Title  
%0 IS BOOK_Preref  
FDEND
```

Result

`BOOK_Author` then refers to the field *au* in the FACS database *BOOKS*, etc.

All current model application adapls use this declaration type.

Example 3

For adapls that are started from an Adlib application, the actual name and location in the file system of the host computer can also be specified in the application structure file (*.pbk*) with Adlib Designer, although this method is now deprecated. In this case, no file name needs to be specified in the adapl, but the (compiled) adapl does need to be present in the application folder that contains the *.pbk*. In our current model applications we do not use this mechanism anymore.

An example of a FACS file declaration when *BOOK* is already defined in the application setup would be:

```
FDSTART BOOK  
au IS BOOK_Author  
ti IS BOOK_Title  
%0 IS BOOK_Preref  
FDEND
```

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.40 FIELDISREPEATED

syntax

```
result = fieldisrepeated (tag_or_field_name)
```

Arguments

result: integer

tag_or_field_name: text

Meaning

Checks whether the provided field is defined in the data dictionary as a repeated field. By default, a field that doesn't occur in the data dictionary, is a repeated field. *Result* = 1 if the field is repeated, and *result* = 0 if it is not repeated.

Example 1

```
result = fieldisrepeated ('institution.name')
```

Result

Result = 0.

Example 2

```
result = fieldisrepeated ('OB')
```

Result

Result = 1

5.2.41 FIELDLENGTH

syntax

```
result = fieldlength (tag_or_field_name)
```

Arguments

result: integer

tag_or_field_name: text

Meaning

Yields the length of the provided field as defined in the data dictionary. By default, fields that are not defined in the data dictionary, have length 0 (word-wrapping is then enabled).

Example 1

```
result = fieldlength ('object.name.notes')
```

Result

Result = 0.

Example 2

```
result = fieldlength ('creator')
```

Result

Result = 255.

5.2.42 FIELDTYPE

syntax

```
result = fieldtype (tag_or_field_name)
```

Arguments

result: integer

tag_or_field_name: text

Meaning

Yields the data type of the field (only the number of the type) as defined in the data dictionary. By default, fields that are not defined in the data dictionary, have type 1. The different data types have the following numbers:

- 1 TEXT
- 2 LETTERS ONLY
- 3 NUMERIC
- 4 INTEGER
- 5 GENERAL DATE
- 6 DATE (USA) MM/DD/YYYY
- 7 DATE (EUR) DD/MM/YYYY
- 8 DATE (ISO) YYYY-MM-DD
- 9 TIME

- 10 ISBN
- 11 ISSN
- 12 LOGICAL
- 13 ENUMERATE
- 14 APPLICATION
- 15 IMAGE
- 16 TEMPORARY

Example 1

```
result = fieldtype ('object.name.notes')
```

Result

Result = 1.

Example 2

```
result = fieldtype ('acquisition.price.value')
```

Result

Result = 3.

5.2.43 FORMATFIELD

syntax

```
result = FORMATFIELD (tag, occ, format)
```

Arguments

tag: Adlib tag

occ: integer

result, format: text

Meaning

This function applies to fields of the *Image* data type and to fields of the *Application* data type:

Image - For this field type, with the *formatfield* function you format the contents of occurrence number *occ* of Adlib database tag *tag* (an image field) according to the 'image', 'thumbnail' or 'default' format. These three options refer to the *Retrieval path*, *Thumbnail retrieval path* and respectively *Storage path* options on the *Image*

properties tab of the relevant tag in the Application browser. In effect, the formatting merges an incomplete path to a linked image from the image field with the rest of the required path as specified in the field properties. Only with the complete path, the image can be retrieved and printed from within an adapl. In *result*, the complete path to the image will be produced, which image may then be printed via the PRINTIMAGE ADAPL command.

More specific, the three format options do the following:

- 'image' – use *Retrieval path* for formatting, unless that property is empty, in which case *Storage path* must be used;
- 'thumbnail' – use *Thumbnail retrieval path* for formatting;
- 'default' – use *Storage path* for formatting.

Application - For this field type, with the *formatfield* function you format the contents of occurrence number `occ` of Adlib database tag `tag` (an application field) according to the 'default' format. This option refers to the *Storage path* option on the *Application field properties* tab of the relevant tag in the Application browser. In effect, the formatting merges an incomplete path to a linked file from the application field with the rest of the required path as specified in the field properties. Only with the complete path, the file can be retrieved and launched from within an adapl. In *result*, the complete path to the file will be produced, which file may then be opened in its associated software application via the LAUNCH ADAPL command.

More specific, the format option does the following:

- 'default' – use *Storage path* for formatting.

Background

It is possible that in your application the field which will hold the path to a linked image, does not display the full path, yet only the file name of the image, for example. If that is the case, then this has been set up for your application specifically in the *Retrieval path* option on the *Image properties* tab of the selected image field.

Adlib automatically merges the file name from the image field with the *Retrieval path* to obtain the complete path or the URL. With this, the relevant image can be retrieved for display in Adlib or for printing to a Word template. However, printing may also be performed by an output format (print adapl). ADAPL accesses data differently from your Adlib application. In this case, a consequence is that current adapls do retrieve the file name from the image field, but do not merge it with the *Retrieval path*, making it impossible to find and print the image. This is where the `FORMATFIELD` ADAPL function becomes necessary: it merges the file name (or partial path) from an image

field with either the *Retrieval path*, the *Storage path* or the *Thumbnail retrieval path* to form a complete path with which the adapl can find and print the image.

Example 1

```
result = formatfield (FN, 1, 'image')
printimage 70, 7, 10, 15, result
```

Result

The incomplete path to the image in `FN[1]` will be formatted according to the *Retrieval path* field property of `FN`. The full path will then be put in the `result` variable, after which `printimage` can retrieve the image from that location and print it at the indicated co-ordinates, in the maximum dimensions of 10 by 15.

When using the *Retrieval path*, *Thumbnail retrieval path* and *Storage path* options for a certain image field, you must adjust any adapls which print images from this field, by having the incomplete path from the image field formatted with `FORMATFIELD` prior to printing.

5.2.44 GETCOUNTER

Syntax

```
value = getcounter (<fdname>, '<counter_name>')
```

Arguments

value: integer

counter_name: text

fdname: FACS database name

Meaning

This function offers an alternative, advanced approach to automatically numbered fields. The difference is that an automatically numbered field must be set up in the relevant field definition in the data dictionary (specifying the format string and the automatic assignment moment), while `getcounter` uses named counters which do not need to be set up anywhere, of which the retrieved value must be formatted by the adapl, and the assignment moment is determined by the moment the adapl is carried out. `Getcounter` is

applicable only in adapls which operate on an Adlib SQL or Adlib Oracle database.

The first time a specific counter is called, Adlib will create it for you and set its value to 0 before retrieving it. Every next time you call this named counter with the `getcounter` function, it will be incremented with 1.

There is no (user-friendly) way to reset a named counter, to set it to a different value, or to delete it, so take care when testing your new functionality: you could use a differently named counter for test purposes, for instance. Also remember to never call `getcounter` in your adapl, unless all your conditions for correct record entry have been met, to prevent "losing" sequential numbers.

Named counters are global counters - Although you have to provide a FACS name in this function, this does not mean that the specified counter is associated with this FACS database only. Named counters are actually server based and the FACS name you submit is only used to identify the SQL or Oracle server which contains the relevant database. Therefore, there exists only one instance of any named counter for all the tables together in an Adlib SQL or Adlib Oracle database. You must explicitly open your FACS database, before you can use `getcounter`.

Applying the newly retrieved number - The new sequential number which you retrieve with the current function can be applied in different ways: you can specify conditions under which the automatic number must be assigned, for example the condition that certain fields must be empty or filled with a particular value, or the condition that the current storage adapl is called from within a specific dataset. You can then format the number anyway you like before writing the value to a database field.

The ADAPL reserved variable `&E` catches any errors produced by `getcounter`.

Example

Suppose you have an existing storage adapl, associated with the *collect* database. Such an adapl will be executed every time you save a record in this database. As an example, we would now like to create an automatically numbered counter for object numbers, although the target field is not that relevant*.

In this adapl you would then declare a FACS database which points to the *collect* database, and call it `OBJECTS`, for example. Next you have to open the FACS database, using: `open OBJECTS`. Further, we would like our object number counter to be called `objno`, for example; you don't need to declare this identifier anywhere before using it in `getcounter`. All you have to do now to

create an automatic counter in your Adlib SQL or Adlib Oracle database, is call `getcounter`. If you have declared an integer variable called `newno`, for example, the call would look like:

```
newno = getcounter (OBJECTS, 'objno')
```

Result

The `newno` variable will be 0 if `objno` is called for the first time ever. If `objno` is called for the second time, `newno` will be 1, etc.

* If the automatic number has be to entered into a field which is a mandatory field, then take into account that the assignment moment of the value is not there when the record is saved, because Adlib will check if mandatory fields have been filled in, before a storage adapl is executed. If the automatic number really must be assigned at the moment the record is stored, and the target field is currently still mandatory (like the object number field), then it may be worth considering making the field non-mandatory, and using the storage adapl to make sure that the field never remains empty.

5.2.45 GETKEY

Syntax

```
key = GETKEY (code)
```

Arguments

key: integer

code: 0, 1, 2 or 3

Platform

DOS

Meaning

Waits for an ASCII key, function key or function combination and gives the identifying value for it.

`GETKEY (0)` indicates whether a key has been pressed. The function does not wait for a key to be pressed. `key = 1` means that a key has been pressed; otherwise, the result will be 0.

`GETKEY (1)` returns the ASCII value of the key pressed. If the key that

was pressed is either a function key, the edit/cursor key or the **Ctrl** key, the result will be 0. The **Enter**, **Esc**, **Backspace**, **Tab** and **Backtab** keys also give 0 as result.

GETKEY (2) waits for a key to be pressed, and returns the Adlib function code for it. The function codes are shown in the table below. Ordinary ASCII keys return the function code 0. Keys that do not have a valid function or ASCII value give the result -1. **Enter**, **Esc**, **Backspace**, **Tab** and **Backtab** are function keys in Adlib.

GETKEY (3) waits for a key to be pressed and returns an ASCII value or an Adlib function code. If a function code is found, it will be returned as a negative number. If an ASCII value is found, the result will be the positive value 1 (**GETKEY (2)** would then result in -1).

The following table shows the function codes for **GETKEY (2)** under MS-DOS.

Code	Function	Key
1	Cursor up	↑
2	Cursor down	↓
3	Cursor left	←
4	Cursor right	→
5	Home	Home
6	Delete rest of page	Ctrl-C
7	Delete rest of line	Ctrl-L
8	Insert character	Ins
9	Delete character	Del
10	Tab	Tab
11	Backtab	Shift-Tab
12	Backspace	Backspace
13	Insert line	Shift-Ins

14	Delete line	Shift-Del
17	Help	Ctrl-F1
18	Insert line buffer	Ctrl-F2
19	Next field	Ctrl-Tab
20	Previous field	Ctrl-Shift-Tab
21	F1	F1
22	F2	F2
23	F3	F3
24	F4	F4
25	F5	F5
26	F6	F6
27	F7	F7
28	F8	F8
29	F9	F9
30	F10	F10
31	F11	F11/Shift-F1
32	F12	F12/Shift-F2
33	F13	F13/Shift-F3
34	F14	F14/Shift-F4
35	F15	F15/Shift-F5
36	F16	F16/Shift-F6
47	Previous screen	Page Up

48	Next screen	Page Down
49	Confirm action/selection	Enter/Return/↵
51	Cursor to End	End
54	Escape	Esc

Note that this function does not work when you run the `adapl` in which it appears as a stand-alone program, through `adeval.exe`: `getkey` will cause the `adapl` to stop running, and the Windows Task Manager has to be used to close the `adapl`.

When an `adapl` with this function is called from within `Adlib` for Windows, `getkey` always returns zero.

5.2.46 GETVAR

Syntax

```
result = GETVAR(variable)
```

Arguments

```
result, variable: text
```

Meaning

`GETVAR` returns the value of the system variable with the name `variable`. Variables with large content (more than 2 KB) can be used. If the variable does not exist, `GETVAR` will return a null string (").

Example

```
LANGUAGE = getvar('ADLIB_DIR')
```

Result

The variable `LANGUAGE` is filled with the contents of the system variable `ADLIB_DIR`. In the standard application, that will be something like `C:\adlib software\bin`.

See also

`SETVAR`

5.2.47 GOSUB

Syntax

```
GOSUB label
```

Meaning

Execution continues with the instruction at `label`. A label is a number at the beginning of a line. Labels must be unique but can be assigned in any order. Labels and the accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no other labels occur). ADAPL will store the location of the `GOSUB` instruction in memory so it can continue with the next line if there is a `RETURN` instruction in the program text somewhere after the label. The program text between a label and the `RETURN` instruction is referred to as a subroutine. From a subroutine you can call another subroutine. After a `RETURN`, ADAPL always goes back to the line following the last encountered `GOSUB`.

See also

`RETURN`

5.2.48 GOTO

Syntax

```
GOTO label
```

Meaning

Execution continues with the instruction at `label`. This label is a number at the beginning of a line. Labels must be unique but can be assigned in any order. Labels and the accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no other labels occur).

See also

`GOSUB`

5.2.49 HTMLTOISOLATIN

Syntax

```
new_text = htmltoisolatin (string)
```

Arguments

```
string, new_text: text
```

Meaning

Converts an HTML encoded string to isolatin.

Example

```
new_text = htmltoisolatin ('&euml')
```

Result

```
new_text is 'ë'.
```

5.2.50 IF THEN ELSE

Syntax

```
IF expression THEN instruction ELSE instruction
```

OR:

```
IF expression THEN {  
    instruction  
    instruction  
    ...  
} ELSE {  
    instruction  
    instruction  
    ...  
}
```

OR (from 6.6.0):

```
IF expression THEN  
{  
    instruction  
    instruction  
    ...  
}
```

```
}  
ELSE  
{  
    instruction  
    instruction  
    ...  
}
```

The key word `THEN` can be omitted. The `ELSE` part is optional.

Meaning

If the result of the expression is `TRUE`, only the instruction or the instruction block following `THEN` will be carried out. If the expression results in `FALSE`, the instruction or the block following `ELSE` is carried out (if present).

See also

Conditional loops

5.2.51 INCLUDE

Syntax

```
INCLUDE 'file'
```

Meaning

The compiler will compile the `file` and then continue compilation of the current file. The `file` may be preceded by a disk or volume name and a directory path name. Single quotes around the file name are compulsory.

It is not allowed to end this statement with a semi-colon.

5.2.52 INPUT

Syntax

```
text = INPUT(number)
```

Arguments

```
text: text
```

`number: integer`

Meaning

Reads a text of maximum `number` of characters from the keyboard, displays them on the screen in a box at the current cursor position, and returns them to the program following a **Return**. If more than `number` of characters are entered, additional characters will be ignored and you will hear an alarm. The following keys can be used during input: **Delete**, **Backspace**, **Left**, **Right**. Keyboard input is always in overwrite mode. As soon as a **Return** is given, the ADAPL program continues. `INPUT(0)` waits until **Return** is pressed.

Example 1

```
name = INPUT(20)
```

Result

The text (max 20 characters) that the user enters, is saved in the variable `name`.

Example 2

```
locate (4, 5)
display '*** Enter the user's login name ***'

locate (6, 5)
variable1 = input(100)

locate (8, 5)
display '*** Enter the user's password ***'

locate (12, 5)
variable2 = input(100)
```

Result

A dialog box is presented, first asking the user to enter the user's login name, after which it can actually be entered (max 100 characters) and will be stored in `variable1`. Then the password is requested, after which that can be entered as well.

5.2.53 INSTR\$

Syntax

```
position = INSTR$(begin, text, search_key)
```

Arguments

text, search_key: text

position, begin: integer

Meaning

Searches for character string `search_key` in `text` starting at position `begin`. Returns the start position if `search_key` is found, otherwise 0.

Example 1

```
position = INSTR$(4, 'abcdefghij', 'ghi')
```

Result

position is 7.

Example 2

```
position = INSTR$(10, 'abcdefghijklm', 'ghi')
```

Result

position is 0.

See also

RINSTR\$

5.2.54 INT

Syntax

```
number = INT(num)
```

Arguments

number: integer

num: numeric

Meaning

Returns `num` as a whole number, the decimal part having been lost. The + or - sign remains.

Example

```
number = INT(14.52)
```

Result

```
number is 14.
```

5.2.55 IS

Syntax

```
tag IS variable_name
```

Meaning

`is` is used for the definition of database variables in FACS. By `tag` we mean the tag as it occurs in the declared file. The `variable_name` is the name (alias) by which the field is known in this procedure. Allocation of an alias is only necessary if a tag also occurs in another database used in this `adapl`, but is always allowed. ADAPL programs are much more readable if you use aliases, which may comprise up to 32 characters, instead of tag names consisting of two characters. In an ADAPL program, each alias may only have one meaning.

Example 1

Example of a FACS file declaration:

```
FDSTART BOOK
au IS BOOK_Author
ti IS BOOK_Title
%0 IS BOOK_Preref
FDEND
```

Result

`BOOK_Author` then refers to the field `au` in the database `BOOK`, etc.

Example 2

Example of a FACS file declaration if you don't want to assign database variables because you're only using one database in this adapl:

```
FDSTART BOOK  
  
au  
  
ti  
  
%0  
  
FDEND
```

Result

Each declared tag here from the database *BOOK*, can now be accessed in this adapl directly.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.56 ISIN

Syntax

```
stat = ISIN(facs, priref)
```

Arguments

stat, priref: integer

facs: FACS definition

Meaning

Stat is 1 if the `priref` occurs in the dataset called `facs`. Otherwise, stat will be 0.

Example

```
stat = ISIN (BOOK, 2002)
```

Result

stat takes the value 1 if record 2002 occurs in the database `BOOK` and 0 if it does not.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.57 ISISBN

Syntax

```
stat = ISISBN(isbn)
```

Arguments

stat: integer

isbn: text

Meaning

stat will be 1 if isbn is a valid ISBN code (either 10 or 13-digit), and 0 if it is not.

Example 1

```
stat = ISISBN('0-07-027988-8')
```

Result

stat is 1.

Example 2

```
stat = ISISBN('12345')
```

Result

stat is 0.

Example 3

```
stat = ISISBN('978-0-07-027988-9')
```

Result

stat is 1.

5.2.58 ISISMN

Syntax

```
stat = ISISMN(ismn)
```

Arguments

```
stat: integer
```

```
ismn: text
```

Meaning

stat will be 1 if ismn is a valid ISMN code (International Standard Music Number), and 0 if it is not.

Example

```
stat = ISISMN('M-3452-4680-5')
```

Result

```
stat is 1.
```

5.2.59 ISSSN

Syntax

```
stat = ISSSN(issn)
```

Arguments

```
stat: integer
```

```
issn: text
```

Meaning

stat will be 1 if issn is a valid ISSN code, and 0 if it is not.

Example 1

```
stat = ISSSN('0927-2739')
```

Result

```
stat is 1.
```

Example 2

```
stat = I$ISSN('12345')
```

Result

```
stat is 0.
```

5.2.60 ISMODIFIED

Syntax

```
stat = ISMODIFIED(tag[occurrence])
```

Arguments

```
stat, occurrence: integer
```

Meaning

stat is 1 if the field belonging to the tag (or the specific occurrence, if requested) has been modified on the screen. Otherwise, status will be 0.

Example

```
stat = ISMODIFIED (au[1])
```

Result

```
stat is 1 if the first occurrence of the field au has been modified,  
and 0 if it has not.
```

5.2.61 ISOLATINTOHTML\$

Syntax

```
new_text = isolatintohtml$ (string, number)
```

Arguments

```
string, new_text: text
```

```
number: integer
```

Meaning

Converts a `string` of isolatin type `number` to HTML. At present, only isolatin type 1 is supported.

Example

```
new_text = isolatintohtml$ ('élan', 1)
```

Result

```
new_text is '&eacute;lan'.
```

5.2.62 ISOLATINTOUTF

Syntax

```
new_text = isolatintoutf (string)
```

Arguments

```
string, new_text: text
```

Meaning

Converts a `string` of isolatin to Unicode in UTF-8 encoding. This function is for use in for instance XML files.

Example

```
new_text = isolatintoutf ('café')
```

Result

```
new_text is 'café'. Only the memory presentation of the string has changed from isolatin to UTF-8.
```

5.2.63 ISSTOPWORD

Syntax

```
stat = ISSTOPWORD( 'word')
```

Arguments

stat: integer

word: text

Meaning

stat is 1 if the word occurs in the stop table and 0 if it does not.

Example

```
stat = ISSTOPWORD ('the')
```

Result

stat is 1 if the occurs in the stop table. In the standard application, this will very likely be the case. If the does not occur, stat will have the value 0.

5.2.64 JSTR\$

Syntax

```
new_text = JSTR$(text, width, type_justification)
```

Arguments

text, new_text: text

width, type_justification: integer

Meaning

If type_justification is a negative number, text will be left-justified within width characters. If type_justification is zero, text will be centred within width characters. If type_justification is positive, text will be right-justified within width characters. The space preceding or following width characters will be blank-filled.

Example

```
new_text = JSTR$('NAME', 10, 0)
```

Result

`new_text` is `'..NAME...'` (for the sake of clarity, spaces are indicated with full stops).

5.2.65 LAUNCH

syntax

```
result = LAUNCH (file name or URL)
```

Arguments

file name or URL: text or database field

result: integer (see table)

0	Result when the launch function can be executed successfully
2	Result when the file or the URL cannot be found

Meaning

LAUNCH (starting an application) is used to open files with the application which is associated with the file extension.

This command has partly replaced the functionality of the already existing **SYSTEM** function, because there may be some confusion with a number of commands. Sometimes it is not possible to make a distinction between commands and file names. An example of this is: `del my.doc`, which can be interpreted as a file name with the name `'del my.doc'` or as a command to delete the file `'my.doc'`. To make this distinction clear, for the first purpose the **LAUNCH** function is used whereas the **SYSTEM** command can now only be used to execute commands.

(If you want to check whether a file on your computer or network actually exists without opening it, use **SYSTEM** with the **RENAME** parameter instead.)

Example 1

```
LAUNCH ('document.doc')
```

Result

During execution of this function in the `adapl`, Adlib will open the file `document.doc` by launching the program set up in Windows which opens files with the extension `.doc` (usually MS-Word).

Example 2

```
LAUNCH('http://www.adlibsoft.com/default.html')
```

Result

Adlib will start your web browser with this URL (Universal Resource Locator: an Internet address).

See also

OPENURL

SYSTEM

5.2.66 LEFT\$

Syntax

```
result = LEFT$(text, number)
```

Arguments

`text`, `result`: text

`number`: integer

Meaning

Returns the first `number` of characters of `text`. If `text` is shorter than `number` of characters, the `result` will be equal to `text`.

Example

```
result = LEFT$('ABCdefGH', 2)
```

Result

`result` is 'AB'.

See also

RIGHT\$

5.2.67 LEN

Syntax

```
length = LEN(text)
```

Arguments

length: integer

text: text

Meaning

Gives the actual length of text in characters. The newline (`\n`), alert (`\a`) and tab (`\t`) characters are counted as two characters each, so `LEN('sport\t')` returns 7.

In the new C# implementation of `eval`, the newline (`\n`), alert (`\a`) and tab (`\t`) characters are counted as a single character, so `LEN('OK,\n')` returns 4.

Example

```
length = LEN('ABC'+ 'def'+ 'GH')
```

Result

length is 8.

5.2.68 LET

Syntax

```
LET variable = expression
```

Meaning

The assignment instruction assigns the result of an `expression` to a variable or an array element. For `variable`, you fill in the name of a variable or an array element, and for `expression`, the operation of which the result is to be saved in the `variable`.

You can leave out the word `LET` because the compiler also recognizes

the combination `variable = expression` as an assignment instruction.

Example

```
integer Counter
numeric Table[5]
text Order_Status[10]
Counter = 10
Table[2] = Counter + 0.5
Order_Status = 'on order'
```

Result

The first assignment in this short program assigns the value 10 to the integer variable `Counter`. The second instruction adds the value 0.5 to `Counter` and assigns the result to element 2 of the numeric array `Table`. The third assignment assigns the text constant `on order` to the text variable `Order_Status`.

5.2.69 LOCATE

Syntax

```
cursor_status = LOCATE(line, column)
```

Arguments

```
cursor_status, line, column: integer
```

Platform

DOS

Meaning

Moves the cursor to line `line`, column `column` on the screen. The maximum values for `line` and `column` depend on the screen dimensions that have been set (normally 24x80). If the result is successful, `cursor_status` will be 1. If the cursor position exceeds the screen dimensions, `cursor_status` = 0 and the cursor will remain unmoved.

Example

```
cursor_status = LOCATE(1, 1)
```

Result

The cursor moves to the top left-hand corner of the screen and `cursor_status` gets the value 1.

5.2.70 LOCK

Syntax

```
LOCK facsname
```

Meaning

With `LOCK` you can still lock a record that you read in without `UPDATE`.

Example

```
READ address NEXT  
IF customer <> '' THEN {  
  LOCK address  
}
```

Result

If the field to which `customer` refers is empty, the record in the database called `address` is locked.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.71 MAX

Syntax

```
maximum = MAX(number1, number2, number3, ...)
```

Arguments

```
maximum, number1, number2, number3: numeric
```

Meaning

Returns the maximum value from the arguments list.

Example

```
maximum = MAX(14.5, 25.2, 3.4)
```

Result

```
maximum is: 25.2
```

5.2.72 MID\$

Syntax

```
result = MID$(text, position, length)
```

Arguments

```
position, length: integer
```

```
result, text: text
```

Meaning

Returns length of number of characters from text, starting at position. The position is "1-based", so `mid$('tkstrf', 1, 1)` retrieves the first character: 't'.

Example

```
result = MID$('ABcdeFGH', 2, 3)
```

Result

```
result is: 'bcd'
```

5.2.73 MILESTONE

Syntax

```
milestone (window_title, message, counter)
```

Arguments

```
window_title, message: text
```

```
counter: integer
```

Platform

Windows

Meaning

This function shows a window containing:

- a window title (entered in `window_title`)
- a text (entered in `message`)
- a counter (entered in `counter`)

By invoking this function with a different value for `counter` each time, the counter will start running. The function should therefore only be invoked if necessary for the progress of the executed action.

The `milestone` function must be invoked at least twice, as is apparent from the example below.

Example

```
counter = 0
while not (condition) {
    milestone ('Counter 1', 'Counter is on: ', counter)
    counter = counter + 1
}
milestone ('', '', -1)
```

Result

In the loop, the function is called for showing the incremented counter. After the loop, the window is closed by calling the function with two empty text arguments and a negative value for

counter.

5.2.74 MIN

Syntax

```
minimum = MIN(number1, number2, number3, ...)
```

Arguments

```
minimum, number1, number2, number3: numeric
```

Meaning

Returns the `minimum` value in the arguments list.

Example

```
minimum = MIN(14.5, 25.2, 3.4)
```

Result

```
minimum is 3.4
```

5.2.75 MKDIR

Syntax

```
mkdir [<drive>:]<path>
```

or

```
md [<drive>:]<path>
```

Meaning

Create a new directory in your file system. This function can only be used in ADAPL as the parameter in a `SYSTEM` command.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Example

```
stat = SYSTEM('mkdir C:\temp\mynewfolder')
```

5.2.76 MOD

Syntax

```
remainder = MOD(numerator, denominator)
```

Arguments

remainder: integer

numerator, denominator: numeric

Meaning

Returns the remainder of numerator divided by denominator as a whole number.

Example

```
remainder = MOD(29, 3)
```

Result

remainder is 2

5.2.77 NAME\$

Syntax

```
new_name = NAME$(old_name)
```

Arguments

new_name, old_name: text

Meaning

Changes 'surname, first name' to 'first name surname'.

(First name must be separated from surname with a comma)

Example

```
new_name = NAME$('Rijn, Rembrandt van')
```

Result

```
new_name is: Rembrandt van Rijn
```

5.2.78 NDAYS

Syntax

```
days = NDAYS(date)
```

Arguments

```
days: integer
```

```
date: text
```

Meaning

`NDAYS` returns the number of `days` between 1 January 1900 and `date`. A negative number is given if `date` is before 1900. The parameter `date` must be a date in the Julian (yyyy.ddd), European (dd/mm/yyyy), or ISO (yyyy-mm-dd) notation. The USA format is not supported.

If an invalid `date` is entered, `days` gets the value 0, and the system tag `&E` (which indicates whether an operation has been successful) is set to 5. It is therefore a good idea to check the contents of `&E` after carrying out the `NDAYS` function.

The `NDAYS` function only works reliably on dates after september 1752, when a new calendar was introduced.

For `date` you can also use `'today'` or `date$(<code>)`. The system will then automatically fill in the value that applies at the time the ADAPL is executed. It is also possible to combine `'today'` or `date$(<code>)` with a plus or minus sign followed by an integer (e.g. `'today' -10`). This represents the system date plus or minus the number of days indicated.

Example

```
days = NDAYS('05/12/1992')
```

Result

```
days is 33941
```

See also

DATE\$

5.2.79 NULL

Syntax

```
tag[#] = NULL()
```

Arguments

tag: database variable

Meaning

Deletes an occurrence from `tag` (if it isn't part of a data dictionary field group). Subsequent elements all move upwards one position. If no occurrence number subscript `[#]` is specified (where `#` must be replaced by a number), `NULL` removes all occurrences from the `tag`. However, if the `tag` is part of a field group as defined in the data dictionary, removing a specific occurrence means deleting the entire field group occurrence, not just of the indicated `tag` occurrence; and removing all occurrences of the specified `tag` means deleting all occurrences of the entire field group!

If you want to empty the contents of one specific occurrence of a specific `tag` in a data dictionary field group, then you can do this with the assignment: `tag[#]=''`.

Note that you shouldn't use `tag=NULL` assignments if the `tag` is part of a dummy FACS database and you use the `adapl` to print to a Word template: this causes an error during printing. Instead, use `tag=''` to assign an empty value. (A dummy FACS database is a FACS declaration which only contains tags that do not exist in the referenced database: these tags can be used to collect data in some way, usually to be printed to a Word template.)

The round brackets for this function may be omitted. If they are used, they may not contain an argument.

Example 1

```
au[3]=NULL()
```

Result

The third author is deleted from the record.

Example 2

```
au=NULL
```

Result

All authors are deleted from the record.

See also

REPINS

5.2.80 ONCHANGEIN

Syntax

```
stat = ONCHANGEIN(tag[occ])
```

Arguments

stat, occ: integer

tag: database variable

Meaning

Stat will be 1 if tag[occ] has changed since the previous call of ONCHANGEIN. Otherwise, stat will be 0. The occ indicates which iteration of the database field is concerned.

Example

```
if onchangein(au[1]){  
  display 'Data has changed'  
}
```

Result

If the first occurrence of the field has been changed, display the message *Data has changed*.

5.2.81 ONEOJ

Syntax

```
ONEOJ label
```

Meaning

Subroutine 'ON End Of Job'. Carry out the subroutine at `label` at the end of the print job. This label is a number at the beginning of the line with which you want to continue. Labels must be unique but can be assigned in any order. Labels and accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no further labels occur). This command only works in combination with the `PAGE` command.

See also

`PAGE`

`ONSOJ`

`ONSOP`

`ONEOP`

5.2.82 ONEOP

Syntax

```
ONEOP label
```

Meaning

Subroutine 'ON End Of Page'. Carry out the subroutine at `label` after printing a page. This label is a number at the beginning of the line with which you want to continue. Labels must be unique but can be assigned in any order. Labels and accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no further labels occur). This command only works in combination with the `PAGE` command.

See also

PAGE

ONSOJ

ONSOP

ONEOJ

5.2.83 ONSCREEN

Syntax

```
stat = ONSCREEN(tag)
```

Arguments

stat: integer

tag: database variable

Meaning

The parameter `tag` is the name of a database variable. `stat` is 1 if `tag` is present on the current screen, and 0 if not. Only applicable to a screen `adapl`.

Example

```
stat = onscreen(A1)
```

Result

If the tag `A1` is on the screen, the variable `stat` gets the value 1. Otherwise, 0.

5.2.84 ONSOJ

Syntax

```
ONSOJ label
```

Meaning

Subroutine 'ON Start Of Job'. Carry out the subroutine at `label` at the beginning of the print job. This label is a number at the beginning of

the line with which you want to continue. Labels must be unique but can be assigned in any order. Labels and accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no further labels occur). This command only works in combination with the `PAGE` command.

See also

`PAGE`

`ONEOP`

`ONSOP`

`ONEOJ`

5.2.85 `ONSOP`

Syntax

```
ONSOP label
```

Meaning

Subroutine 'ON Start Of Page'. Carry out the subroutine at `label` before printing a page. This label is a number at the beginning of the line with which you want to continue. Labels must be unique but can be assigned in any order. Labels and accompanying program code always come after the last `END` instruction. A block of code belonging to a label continues until the next label or until the end of the program text (if no further labels occur). This command only works in combination with the `PAGE` command.

See also

`PAGE`

`ONSOJ`

`ONEOP`

`ONEOJ`

5.2.86 OPEN

Syntax

```
OPEN facsname  
OPEN facsname NORESET  
OPEN facsname POINTER = number
```

Arguments

facsname: facsname
number: integer

Meaning

Opens the file that you declared as facsname earlier in the procedure. The index and record pointers are moved to the starting position. Adlib will initialize all database variables for this file to null. The file and accompanying variables then become available for FACS operations.

With the `OPEN` instruction the system variable `&E` is set to an error code. The value 0 means that the opening procedure was successful. Any other value means that an error occurred. The meaning of those values can be found in the error code list.

If the file has already been opened, Adlib will not open it again, but just resets it in memory, unless you use `OPEN ... NORESET`. In that case, the index and record pointers will remain unchanged.

To use the selection in a pointer file when reading from a file, use `OPEN ... POINTER` and enter the number of the pointer file you want to use when opening the file (see example).

Example

```
open BOOK pointer = 1
```

Result

The search statement in the pointer file with the name 1 is used to read from the previously declared facsname `BOOK`.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.87 OPENFILE

Syntax

```
handle = openfile ('path',0)
```

Arguments

handle: integer

path: text

Meaning

OPENFILE opens the text file named in `path`. If opening is successful, `handle` will assume a value of 0 or a positive value. If opening is not successful, `handle` takes the value -1. `handle` is then used as a unique identification for the corresponding text file. The second parameter (0) is reserved for future use.

Several text files can be opened in one ADAPL, provided a different handle is used for each text file.

The name of a text file may be preceded by a path. If there is no path in front of the text file, then the file is sought in the working directory. If it is not found there, the ADAPL will start looking in the directory to which the environment variable ADLIB_DIR refers. In this way, you can also use general Adlib text files in your ADAPL.

See also

Handling text files

5.2.88 OPENURL

Syntax

```
result = openurl (text, modus)
```

Arguments

text: text (string, string variable, or database field/tag containing the URL to be opened)

modus: 0 or 1

result: numeric

Meaning

With this function you can only open internet addresses; addresses on your local network or workstation cannot be opened with `openurl`.

`Openurl` with `modus=1` opens the entered URL in a browser window.

With `modus=0`, `openurl` checks whether the URL in a field is valid. The resulting numerical value conforms to the W3C http standard; on successful opening of the URL, *200* is returned, while failure returns *404*. Besides that, there are dozens of other result codes.

The modes 0 and 1 should be used successively. First, with `modus=0`, check whether the URL can be opened, and only when `result=200` you should actually open the URL with `modus=1`.

This function can also be used in stand-alone adapls.

(If you want to check whether a file on your computer or network actually exists without opening it, use `SYSTEM` with the `RENAME` parameter instead.)

Example

```
IF openurl('http://www.adlibsoft.com/', 0) = 200 THEN {  
  openurl('http://www.adlibsoft.com/', 1)  
}
```

See also

LAUNCH

SYSTEM

5.2.89 OUTPUT

Syntax

OUTPUT LINE

OUTPUT PAGE

OUTPUT SPOOL

OUTPUT SKIP number

Arguments

number: integer

Meaning

Sends the line built up by print instructions to the print file. Following the `OUTPUT` instruction, the print buffer is cleared.

`LINE` adds a carriage return to the buffer and adds 1 to the line count (&L).

`PAGE` adds a form feed, adds 1 to the page count (&S) and resets the line count (&L) to 0.

`SPOOL` closes the print file and sends it to the printer. After an `OUTPUT` `SPOOL` the print file is cleared, and &L and &S are reset to 0.

With `SKIP number` you can indicate that a number of empty lines are to be printed. Adlib will add the entered number to the line count (&L).

5.2.90 PAGE

Syntax

```
PAGE (y, x)
```

Arguments

y, x: integer

Meaning

Indicates the size of the printable area on a page: height (y) and width (x). Dimensions for this function are in AUs (Adlib Units). A horizontal AU is 1/90 of the width of a standard printable A4 area; a vertical AU is 1/72 of the length of the standard printable A4 area. (The standard font and font size for printing exactly 90 characters across the full available page width, is Courier size 11. If you change the margins yourself, then correspondingly the number of units that can be printed will be reduced. If you change the font you will be able to print fewer or more characters across the width of the page, depending on the font and font size you choose.)

Whenever the `PAGE` command is used in an `adapl`, 2 AUs of the specified height will be reserved for both the header and footer on every page (so, 4 AUs per page in total).

5.2.91 PAGEBREAK

Syntax

```
PAGEBREAK (ASCII value)
```

Arguments

```
ASCII value: integer
```

Meaning

The ASCII value given to `pagebreak` stands for the form feed character of the printer. When the printer reaches this character, it does not print it but goes to a subsequent page. `PAGEBREAK (0)` suppresses the form feed.

5.2.92 PDEST

Syntax

```
PDEST 'name'
```

```
PDEST 'name' FILE
```

```
PDEST 'AUXPORT'
```

```
PDEST 'pause_text' SCREEN
```

Arguments

```
name, pause_text: text
```

Meaning

Indicates that the results of print jobs are to go to a printer with a specified `name`. This may be `LPT1` or `LPT2` for stand-alone computers, or a path to a printer in a network.

The option `FILE` indicates that the destination is a file and not a printer.

The option `AUXPORT` indicates that printing is to take place via the printer port of the terminal.

The option `SCREEN` makes it possible to print to the screen from stand-alone adapls. As `pause_text` you enter a text that is to appear when

the screen is full and the adapl must await your okay to show the next page. E.g.: *Press any key...* (it may also be empty, in which case everything will be sent to the screen at once).

5.2.93 PRINT

Syntax

```
PRINT pos, max, expression
```

Arguments

pos, max: integer

expression: text

Meaning

The `PRINT` instruction places the text resulting from the expression in the print line, starting at position `pos` and with a maximum length `max`. If `max` is not enough to print everything, the text is automatically word-wrapped. The line is built up by `PRINT` instructions and closed with an `OUTPUT` instruction.

Dimensions for this function are in AUs (Adlib Units). A horizontal AU is 1/90 of the width of a standard printable A4 area; a vertical AU is 1/72 of the length of the standard printable A4 area.

The standard font and font size for printing exactly 90 characters across the full available page width, is Courier size 11. If you change the margins yourself, then correspondingly the number of units that can be printed will be reduced. If you change the font, you will be able to print fewer or more characters across the width of the page, depending on the font and font size you choose.

The number of lines printed by adapls depends partly on the type of printer you are using. Test how many lines your printer will print in the specified font and use that number as the default for printing.

See also

COLUMN

OUTPUT

PAGE

PRINTIMAGE

5.2.94 PRINTIMAGE

Syntax

```
printimage x, y, width, height, tag_or_path
```

Arguments

`x`, `y`, `width`, `height`: integer

`tag_or_path`: tag or text

Meaning

With `PRINTIMAGE` you can print an image anywhere on a page, along with text (for which you use `PRINT` and `OUTPUT`) if required. You determine the maximum size of the printed image (`width` and `height`) and its location on the page with respect to the upper left corner (`x` and `y`). Choose an empty location on the page to print the image and make sure texts will not be printed there too. The aspect ratio of the image will remain intact no matter what dimensions you provide, so the space between two images might not always be as you expected (but possibly bigger).

`Tag_or_path` should hold the URL to an image, or the path to an image (between quotes).

Dimensions for this function are in AUs (Adlib Units). A horizontal AU is 1/90 of the width of a standard printable A4 area; a vertical AU is 1/72 of the length of the standard printable A4 area.

`PRINTIMAGE` needs no `OUTPUT` to send the image to the printer.

Note that the user can't print images to a file via a print adapl, instead of printing directly to the printer, because it generates a simple text file. For such a purpose you should use Word print templates.

Example

```
printimage 70, 7, 10, 15, 'c:\MyImage.jpg'
printimage 70, 25, 10, 15, B1[1]
```

Result

The image located in `c:\` and the image linked to in the first

occurrence of B1 (image reference field *Identifier URL*) are printed at the same size (10x15 AUs), 3 AUs above each other.

5.2.95 PROGRESS

Syntax

```
progress (title, message, min, max, step)
```

Arguments

```
title, message: text
```

```
min, max, step: integer
```

Platform

Windows

Meaning

This function shows a window containing:

- a window title (entered in `title`)
- a text (entered in `message`)
- a progress bar (the minimum and maximum values are determined by `min` and `max`, and the step size is entered in `step`)

The `progress` function must be called at least twice, as will be apparent from the example below. The progress bar increases in size each time it is called. You are therefore advised to only call this function when necessary for the progress of the executed action.

Example

```
while not (condition) {  
  progress ('Progressbar', 'Progress: ', 0, 100, 2)  
}  
progress ('', '', 0, 0, -1)
```

Result

In the loop, the function is called that displays the progress bar, which has increased in size. After the loop, the window is closed

by calling the function with two empty text arguments, the value 0 for both `min` and `max`, and a negative value for `step`.

5.2.96 QUIT

Syntax

```
QUIT
```

Meaning

The compiler stops compiling. Any errors are stated, but there is no executable program.

5.2.97 READ

Syntax

```
READ facsname  
READ facsname NEXT  
READ facsname UPDATE  
READ facsname USING index  
READ facsname POINTER
```

Meaning

The `READ` operation looks for a record in `facsname` and reads it into memory. After a read operation the error status variable `&E` contains the status code 0 (when successful) or any other number (if unsuccessful).

If the `READ` operation is successful (status code = 0) all variables that have been declared for the file receive the values from the retrieved FACS record. The location in the file or in the index used is saved for a subsequent read operation.

If a `READ` operation is not successful (status code \neq 0) the variables for that external file will not be defined.

The file must be opened with `OPEN` before the `READ` operation can be

carried out.

Options

`READ` without options reads the first record in the FACS dataset. `READ` instructions with options can be combined if necessary.

- **read...next**

The `READ facsname NEXT` operation uses the current position to find the next record. The first `READ NEXT` after opening a dataset is equivalent to a normal `READ` operation.

Warning: `READ...NEXT` is often used in a loop which also, occasionally, writes changes to a record (or deletes a record) which has been read, and at the end of the loop reads the next record. This construction works fine in CBF databases but not in an Adlib SQL or Adlib Oracle database. The solution to this problem is to use a double FACS declaration for the same Adlib database: reading and reading the next record must then be done in FACS declaration 1, while writing or deletion must be done in FACS declaration 2. To get the same record in FACS 2 as already read with FACS 1, use `READ...USING` with the current `pref`. See example 2 below for a clarification.

- **read...update**

If a record is found, `READ facsname UPDATE` locks it so that you are the only one who can carry out a `DELETE` or `WRITE` operation.

Other users can then read, but not lock the record. If no record is found, no lock will be set.

Once set, a lock must be removed and this will happen when you execute a `DELETE`, `WRITE`, or `UNLOCK` instruction. No one else can edit the record as long as the lock remains.

- **read...using**

In `READ facsname USING variable = search_key` the variable should refer to a valid field index for the database. If the indexed field is declared with an alias (using `IS`), you must use the alias. If your `search_key` is literally a text, put quotes around it. You can use a truncated search (on literal text) by ending the search key string with `'*`.

Note that an index for the searched field is not required per se, but it does make `READ USING` much faster.

The key value is searched in the index at `variable`. If `search_key` occurs, the corresponding record is loaded into memory, and the index location is registered. If the `NEXT` option is used, the next record meeting the requirement will be retrieved.

A loop containing `READ ... NEXT UPDATE USING index = key` and

a `WRITE` or `DELETE` instruction is dangerous because the keys for the current record in the index table can either be moved or deleted, thus making the pointer to the index table invalid.

While in this kind of loop, you should not use the `DELETE` command on the FACS file, and you should not modify the index field on which you are searching.

`READ` options may be combined freely, as exemplified in example 1. It reads and locks from an address file all the names starting with 'A', then writes a sequential number in each record.

read...pointer

You can also search for records using the selection in a pointer file. To do this, open the FACS file with the pointer option and use the command `READ facsname POINTER` to read the first record, or `READ facsname NEXT POINTER` and Adlib will give the next record referred to in the pointer file.

Example 1 (for CBF databases only)

```
* PROGRAM 'NUMBER_ADDRESSES'
* Sequential numbers with names
*
* LOCAL DECLARATIONS

text Name[10]
integer Counter

*
* FILES
*

fdstart ADDRESS
  %1 is ADDRESS_Name /* Name
  A1 is ADDRESS_Order /* Serial number
fdend

*
* INITIALIZE
*

Name = 'A*'
Counter = 1

open ADDRESS
if (&E <> 0) {
  end
```

```

}
*
* SEARCH, LOCK and WRITE
*
DO { /* Repeat..
    read ADDRESS next update using ADDRESS_Name = Name
    /* read and lock
    if &E = 0 { /* found one?
        ADDRESS_Order = Counter /* counter as string
        Counter = Counter + 1 /* increment
        write ADDRESS /* write and unlock
    }
} UNTIL &E <> 0
*
* FINISH
*
close ADDRESS /* done
end

```

Example 2 (for CBF and SQL databases)

* Open all records and save them, so that storage adapls
* and such are executed.
* The double FACS declaration is needed only for SQL but
* works in CBF as well, because a READ NEXT after writing
* or deleting a record is problematic in SQL.

```

fdstart COLLECT '../data+collect'
    %0 is recpriref
fdend

```

```

fdstart COLLECT2 '../data+collect'
    %0 is recpriref2
fdend

```

```

open COLLECT
if (&E) {

```

```

        errorm 'Error opening COLLECT: ' + &E
    }

    open COLLECT2
    if (&E) {
        errorm 'Error opening COLLECT2: ' + &E
    }

    read COLLECT
    while (&E = 0) {
        read COLLECT2 using recpriref2 = recpriref
        lock COLLECT2
        write COLLECT2
        if (&E <> 0) {
            errorm 'Error writing COLLECT2 record number: ' +
recpriref + ', error code: ' + &E
            unlock COLLECT2
        }
        read COLLECT next
    }
end

```

See also

[Click here for more information about how to define a FACS name in the application setup.](#)

[Click here for general information about FACS.](#)

5.2.98 RECCOPY

Syntax

```
stat = RECCOPY(sourcefacname, destinationfacname)
```

Arguments

stat: integer

sourcefacname, destinationfacname: facsname

Meaning

Copies the contents of the record that was read in `sourcefacname` to the record that was read in `destinationfacname`.

The record number of `destinationfacname` remains unchanged. The current contents of `destinationfacname` is overwritten with the contents of `sourcefacname`.

For `RECCOPY` to be effective, `destinationfacname` must be written with `WRITE destinationfacnaam` or `WRITE destinationfacname NEXT`. If `WRITE NEXT` is used, a new record is added in `destinationfacname`.

See also

[Click here for more information about how to define a FACS name in the application setup.](#)

[Click here for general information about FACS.](#)

5.2.99 RECDATE\$

Syntax

```
date_time = reccdate$(facname,code)
```

Arguments

`date_time`: text

`facname`: facname

`code`: see table below

Meaning

Gives the input date or time or mutation date or time of the current record (opened with FACS) in `date_time`. The table below shows which code you must use to retrieve the various data.

Code	Meaning
1	mutation date
2	mutation time

3	input date
4	input time

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.100 REDISPLAY

Syntax

```
REDISPLAY
```

```
REDISPLAY NOEXEC
```

```
REDISPLAY CONTINUE
```

Meaning

`REDISPLAY` redisplay the current screen. This is useful if the user was switching from the current screen to another, while some work on the current screen was not yet finished (correctly). The `REDISPLAY` instruction only works in adapls that are linked to a screen. Any changes made by the adapl are displayed. The `REDISPLAY` instruction is not executed immediately: a flag is set and when adapl execution has ended the actual redisplay is performed.

The `REDISPLAY CONTINUE` instruction causes an earlier `REDISPLAY` instruction to be cancelled (the mentioned flag is reset) so that execution of the adapl and the switch to another screen and field continues as if no `REDISPLAY` was issued.

The `NOEXEC` option prevents the adapl from being carried out again for the current record.

5.2.101 REGVALIDATE

syntax

```
result = regvalidate (expression, string)
```

Arguments

result: integer

expression, string: text

Meaning

Validates a `string` to a so called regular expression. *Result* = 0 if the string is formatted according to the provided regular expression, and *result* = 1 if the `string` does not comply to it.

Example 1

```
result = regvalidate ('[0-9]*', '123')
```

Result

Result = 0.

Example 2

```
result = regvalidate ('[0-9a-b]*', '12a#4b')
```

Result

Result = 1 (because # cannot be used, according to the provided regular expression).

5.2.102 RENAME

Syntax

```
rename [<drive>:][<path>]<file name 1> <file name 2>
```

or

```
ren [<drive>:][<path>]<file name 1> <file name 2>
```

Meaning

Change the name of a file (file 1) or of several files at once, into file

name 2. You have to use existing drives and paths (relative, local or UNC paths). File name 2 cannot be preceded by a path. This function can only be used in ADAPL as the parameter in a SYSTEM command.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Besides its intended use, `RENAME` can be used as a work-around to check if a file is present somewhere, without having to try to open the file. You can do that by renaming the file to the name it already has, so effectively the file remains unchanged but the result code from `SYSTEM` will tell you indirectly if the file exists. If the result is 0 the "renaming" was successful, so the file most probably exists (unless it is opened for editing somewhere), and if the result is anything other than 0, the file most probably doesn't exist.

Note that you cannot use Adlib variables or tags as arguments for `RENAME`, but you *can* use an Adlib variable as the argument for `SYSTEM`. So if you want to use variable file names (maybe coming from an Adlib field), you'll have to build up the entire `SYSTEM` argument string in a variable and pass the variable to `SYSTEM`, for example:

```
argumentstringvar = 'rename ' + pathplusfilename1var + ~
' ' + filename2var
resultvar = system(argumentstringvar)
```

Example 1

```
stat = SYSTEM('rename test.txt fail.txt')
```

Example 2

`RENAME` can be used as a work-around to check if a file is present somewhere, without having to try to open the file. A partial code sample is the following:

```
* Find the last backslash in the path taken from a field
slashposfromback = rinstr$(len(pathplusfilenamefield[i]),
pathplusfilenamefield[i] , '\')

* Calculate the position of that backslash from the left
slashposition = len(pathplusfilenamefield[i]) - ~
slashpositionfromback

* Extract the file name from the full path
filename = right$(pathplusfilenamefield[i], slashposition)

* Put the entire SYSTEM argument string together for
```

```

* renaming the file to itself
fileteststring = 'rename ' + pathplusfilenamefield[i] + ~
                ' ' + filename

* Execute the SYSTEM command with the argument string
filetest = system(fileteststring)

* If not successful, the file probably doesn't exist.
* Show error message.
if (filetest) {
    errorm 'The file ' + pathplusfilenamefield[i] + ~
          ' doesn''t exist yet.'
}

```

5.2.103 REPCNT

Syntax

```
number = REPCNT(tag)
```

Arguments

number: integer

tag: database variable

Meaning

If `tag` is not part of a data dictionary field group, `REPCNT` counts the number of occurrences of the `tag`, up to and including the last filled occurrence; trailing empty occurrences don't exist in the database. If `tag` is part of a data dictionary field group, `REPCNT` counts the maximum number of occurrences of any field in the field group, up to and including the last filled occurrence.

Example

```
number = repcnt (A1)
```

Result

`number` takes as value the number of occurrences of the tag `A1`.

5.2.104 REPCOPY

Syntax

```
stat = REPCOPY(tag1, tag2)
```

Arguments

stat: integer

tag1, tag2: database variable

Meaning

Copies all occurrences of tag1 to tag2. stat is 1 if the operation is successful, otherwise 0.

If tag2 is part of a data dictionary field group, then all fields from that field group will be emptied before tag1 is copied. It doesn't matter whether tag1 is part of a data dictionary field group or not: only tag1 will be copied, not the other fields in the group (with REPCOPY you cannot specify a group field mapping).

Example

```
stat = repcopy (au,ca)
if (stat = 1) THEN display 'Copy successful'
```

Result

If the value of stat is 1, the message *Copy successful* will be printed.

5.2.105 REPFIND

Syntax

```
stat=REPFIND(tag,position,text,type_search)
```

Arguments

stat, position: integer

tag: database variable

text: text

type_search: integer, see table below

Meaning

Searches the occurrences of `tag`, starting at occurrence `position`, for text `text`. Returns the number of the occurrence in which `text` was found. If `text` has not been found, the function will return a 0.

With `type_search` you determine the type of data that you want to search on:

Type_search	Meaning
0	Search case- and accent sensitive on text
1	Search on number
2	Search on date
256	Search case insensitive on text
1024	Search accent insensitive on text
1280	Search case- and accent insensitive on text

Example

```
repfind(au, 3, 'Bloggs, J.', 0)
```

Result

The text `Bloggs, J.` is sought from the third occurrence of tag `au`. Names like "Bloggs, j." or "Blógg, J." will not be retrieved.

5.2.106 REPINS

Syntax

```
stat = REPINS(tag, position, text)
```

Arguments

`stat`, `position`: integer

`tag`: database variable

`text`: text

Meaning

Inserts `text` as occurrence `position` of database variable `tag`. All subsequent occurrences will move down to accommodate the new entry. If the action is successful, `stat` will be 0, otherwise unequal to 0.

If the `tag` is part of a data dictionary field group, an empty occurrence at the same position will automatically be created for each of the other fields in the group, so that field group occurrences stay together. If you want to fill any of the empty occurrences of the new group occurrence, simply use `tag[#]=value` assignments, not `repins` (since `repins` creates a whole group occurrence).

Example

```
repins(%1, 3, 'Bloggs, J.')
```

Result

The text `Bloggs, J.` is inserted as the third occurrence of tag `%1`.

NB: Inserting an occurrence into a field in a screen field group leads to the corresponding occurrences of various fields no longer appearing together on the screen. This problem does not occur when the field group is defined as a group in the data dictionary (as opposed to on a screen). Having defined a group both on screen and data dictionary level may sometimes cause conflicts, so make sure field groups are only defined in the data dictionary, not on screen level.

See also

NULL

5.2.107 REPLACE\$

Syntax

```
result = replace$ (startpos, string_or_tag,  
string_to_be_replaced, string_to_be_entered, options)
```

Arguments

```
result, string_or_tag, string_to_be_replaced,  
string_to_be_entered: text
```

startpos: numeric

options: 1, 2, 4, or 8 (see table below)

Meaning

This function replaces a (series of) character(s) by another character or series of characters in a string or tag, and searches towards it from character number `startpos`.

options (numeric):

1	Replace all occurrences of <code>string_to_be_replaced</code> .
2	Ignore uppercase and lowercase distinctions.
4	Replace whole words only.
8	Replace <code>string_to_be_replaced</code> only if it fills the entire field.

You can run several options at once by adding them and just entering their sum as one option.

Examples

```
result = replace$ (1, U1, '\\', '/', 1)
/* replaces all backslashes with forwardslashes in field U1;

result = replace$ (1, TI[1], 'The', '', 6)
/* replaces 'The' as a whole word with nothing, and ignore case;

result = replace$ (1, SA, '\n', '<br>', 1)
/* replaces \n (new line) with an HTML break-tag
```

5.2.108 REPMAX

Syntax

```
maximum = REPMAX(tag)
```

Arguments

maximum: numeric

tag: database variable

Meaning

Returns the occurrence contents with the highest value from all occurrences of database variable `tag`.

Example

```
NN[1] = 5.4  
NN[2] = 7  
NN[3] = 2  
maximum = REPMAX(NN)
```

Result

maximum is 7.

5.2.109 REPMIN

Syntax

```
minimum = REPMIN(tag)
```

Arguments

minimum: numeric

tag: database variable

Meaning

Returns the occurrence contents with the lowest value from all occurrences of database variable `tag`.

Example

```
NN[1] = 5.4  
NN[2] = 7  
NN[3] = 2  
minimum = REPMIN(NN)
```

Result

minimum is 2.

5.2.110 REPSORT

Syntax

```
stat = REPSORT(tag, sort_mode)
```

Arguments

stat: integer

tag: database variable

sort_mode: see tables below

Meaning

Sorts the occurrences of tag. Using `sort_mode` you can indicate how Adlib is to sort.

Sort_mode	Meaning
0	text sort
1	numeric sort
2	date sort

By adding a value to the `sort_mode` code, you can specify in greater detail how you want to sort.

Sort_mode	Meaning
sort + 8	sort in descending rather than ascending order
sort + 256	in text mode: as upper case
sort + 512	in text mode: as lower case
sort + 1024	in text mode: ignore accents

You can combine these extra values. If `sort_mode` is equal to 1288 (8 + 256 + 1024) that means: sort on text, as capitals, and ignore

accents while sorting.

Example

```
stat = REPSORT (au,1024)
```

Result

After a text sort of all authors, with the accents being ignored, `stat` is 0 (if the sort has been successful).

NB: Carrying out a sort on a field in a screen field group leads to the corresponding occurrences of various fields no longer appearing together on the screen. This problem does not occur when the field group is defined as a group in the data dictionary (as opposed to on a screen). Having defined a group both on screen and data dictionary level may sometimes cause conflicts, so make sure field groups are only defined in the data dictionary, not on screen level.

5.2.111 REPSORTINS

Syntax

```
stat = REPSORTINS (tag, text, sort_mode)
```

Arguments

`stat`: integer

`tag`: database variable

`text`: text

`sort_mode`: see tables below

Meaning

Inserts `text` into a new occurrence of `tag`, after which all occurrences of this tag, including the new occurrence, will be sorted. With `sort_mode` you can indicate how Adlib is to sort.

Sort_mode	Meaning
0	text sort
1	numeric sort

2	date sort
---	-----------

By adding a value to the `sort_mode` code, you can specify in greater detail how you want to sort.

Sort_mode	Meaning
sort + 8	sort in descending rather than ascending order
sort + 256	in text mode: as upper case
sort + 512	in text mode: as lower case
sort + 1024	in text mode: ignore accents

You can combine these extra values. If `sort_mode` is equal to 1288 (8 + 256 + 1024) that means: sort on text, as capitals, and ignore accents while sorting.

Example

```
stat = REPSORTINS(au, 'Tuck, F.', 1024)
```

Result

After inserting a new occurrence with the text 'Tuck, F.', a text sort of all authors, with the accents being ignored, `stat` is 1 (if the sort has been successful).

NB: Carrying out an occurrence insert plus sort on a field in a screen field group leads to the corresponding occurrences of various fields no longer appearing together on the screen. This problem does not occur when the field group is defined as a group in the data dictionary (as opposed to on a screen). Having defined a group both on screen and data dictionary level may sometimes cause conflicts, so make sure field groups are only defined in the data dictionary, not on screen level.

5.2.112 REPSUM

Syntax

```
total = REPSUM(tag)
```

Arguments

total: numeric

tag: database variable

Meaning

Returns the sum of the values from all occurrences of database variable `tag`. The field doesn't need to be an integer or numerical field per se: it can be a text field as well, but then make sure that the field doesn't contain text other than numbers and that fractional numbers use a dot as the decimal separator, not a comma.

Example

```
NN[1] = 5  
NN[2] = 7.3  
NN[3] = 2.05  
total = REPSUM(NN)
```

Result

```
total is 14.35
```

5.2.113 RETURN

Syntax

```
RETURN
```

Meaning

Instructs the ADAPL program to continue execution at the line following the last encountered `GOSUB` instruction.

See also

GOSUB

5.2.114 RIGHT\$

Syntax

```
substring = RIGHT$(text, number)
```

Arguments

text, substring: text

number: integer

Meaning

Returns the right part of a character string. If `number` is greater than length of `text`, the result will be equal to `text`.

Example

```
substring = right$('ABCdefGH', 3)  
substring2 = right$('AFDFD', 0)
```

Result

substring is 'fGH'.
substring2 is empty.

See also

LEFT\$

5.2.115 RINSTR\$

Syntax

```
position = RINSTR$(start, text, search_key)
```

Arguments

position, start: integer

search_key, text: text

Meaning

Searches for the character string `search_key` in `text`, starting at position `start` and working back to position '1' in `text`. Returns the start position if found, otherwise '0'.

Example

```
position = rinstr$(11, 'ABCDefGHBCefg', 'BC')
```

Result

position is 9.

See also

INSTR\$

5.2.116 RMDIR

Syntax

```
rmdir [/S] [/Q] [<drive>:]<path>
```

or

```
rd [/S] [/Q] [<drive>:]<path>
```

Meaning

Delete a directory from your file system. This function can only be used in ADAPL as the parameter in a SYSTEM command.

`/S` not only removes all subdirectories and files, but also the directory itself.

`/Q` in combination with `/S`, removes a directory structure without asking for confirmation.

Parameters between [and] are optional, and parameters between < and > must be replaced by an actual value. Do not include any of these brackets in your command.

Example

```
stat = SYSTEM('rd /S C:\temp\mynewfolder')
```

5.2.117 ROUND

Syntax

```
result = ROUND(number, precision)
```

Arguments

```
result, number: numeric
```

```
precision: integer
```

Meaning

Returns the value of `number` rounded off to `precision` number of decimal points. The method of rounding is: 5 or more: upwards; below 5: downwards.

Example

```
result = round(1200.125, 2)
```

Result

```
result is 1200.13
```

5.2.118 ROUND\$

Syntax

```
result = ROUND$(number, precision)
```

Arguments

```
result: text
```

```
number: numeric
```

```
precision: integer
```

Meaning

Returns the value of `number` rounded off to `precision` number of decimal points, as a string. The method of rounding is: 5 or more: upwards; below 5: downwards.

Example

```
result = round(1200.125, 2)
```

Result

```
result is '1200.13'
```

5.2.119 SENDMAIL

Syntax

```
call = sendmail(from-address, to-address, cc-address,  
subject, message-body, file-attachments, format)
```

Arguments

```
from-address, to-address,  
cc-address, subject,  
message-body,  
file-attachments: text
```

```
format: 1 or 0
```

```
call: integer
```

Platform

Windows

Meaning

Sends an e-mail to the specified recipient(s). `from-address` is a mandatory e-mail address of the sender; `to-address` is a mandatory e-mail address of the recipient, `cc-address` is an optional e-mail address of a secondary recipient; `subject` is optional; `message-body` is the actual message; `file-attachments` is an optional list of filenames. `to-address`, `cc-address` and `file-attachments` may be repeated with semi-colon separators. The `format` argument is optional: 1 means that the message body must be interpreted as an HTML page (see the `TRANSFORM` function), while 0 means that the message body is plain text; plain text is the default setting if the `format` argument is left out (so that the `sendmail` statement has only six arguments).

The function operates either with SMTP or MAPI (Microsoft's messaging application programming interface), although the `format 1` only works with SMTP. If a so-called `smarthost` environment variable has been set, SMTP will be used, otherwise MAPI will be attempted,

but with MAPI the sender address needs to be the sender address of the currently logged-in user.

Note that you always need to provide at least six arguments (and seven at most); only for said optional arguments among the first six you may enter an empty value. You do this with two single quotes ". Literal e-mail addresses and text need to be enclosed by single quotes too. The last argument (*format*) and the comma in front of it, may be left out altogether.

Call is zero if `sendmail` was executed successfully. Each other value is an Adlib error code. If any of the mandatory arguments is missing, an error code is generated and there won't be a different attempt to send the e-mail. If the `file-attachments` argument is provided, but the file cannot be found, then the e-mail won't be sent and *Call* will be 11.

A special option is to use the contents from a text file as the message body, without having to specify the name of this file: a `sendmail` command with an empty `message-body` argument (two single quotes) will automatically use the contents from a text file if that text file has been generated in the current `adapl`. Typically you would code this as follows:

```
pdest 'myfilename.txt' file
print 1, 0, 'This is the first line of the message body'
output
print 1, 0, '...'
output
print 1, 0, 'This is the last line of the message body'
output
sendmail(sender, receiver, '', subject, '', '')
```

However, this functionality is not self-evident to anyone reading your code, so it would be good to document it in the code. Also note that if you are not going to use the text file for anything else, there's a better way to build up the text for a message body: just declare a long text variable (like `text mailbody[0]`), add text strings and carriage returns plus line feeds as shown below and enter the `mailbody` variable as the fifth argument in your `sendmail` command.

```
mailbody = 'This is the first line of the message body' +
CHR$(13) + CHR$(10) + ~
          '...' + CHR$(13) + CHR$(10) + ~
          'This is the last line of the message body'
sendmail(sender, receiver, '', subject, mailbody, '')
```

Setting ADLIB_SMARTHOST

For sending e-mail through SMTP, the variable `ADLIB_SMARTHOST` must be set to the name of the mail server which Adlwin should use. With ADAPL, this environmental variable can be set as follows: enter the line `setvar('ADLIB_SMARTHOST', '<name-of-my-smtp-server>')` in the ADAPL file from which e-mails are sent (and replace `<name-of-my-smtp-server>` by the actual name of your smtp server), preferably at the start of the file or at least before an e-mail is sent. You can only send out e-mails from within the adapl with this code!

Example

```
setvar('ADLIB_SMARTHOST', 'mailserver.ourmuseum.com')
```

5.2.120 SETFONT

Syntax

```
code = setfont ('fontname style', pointsize)
```

Arguments

code: integer

fontname, style: text

pointsize: numeric

Platform

Windows

Meaning

Sets the font which is to be used for printing, to `fontname` (font names are case-sensitive) with a particular `pointsize`. You can use proportional (like Arial) as well as non-proportional fonts (like Courier). Behind the font name but still within the single quotes, you may also add any of the following styles or combinations of them: *bold*, *italic*, and/or underline. Which fonts are available depends on the font collection installed in Windows and on the capabilities of your printer.

A printout may look different on various printers. This is mainly caused by printer properties which cannot be influenced by the Adlib software. You may for instance have selected a font and font size in the adapl, that are unavailable in the printer. In such cases, Adlib first tries to print in the specified font but in another font size, and if that

does not work, it will select a different font. So preferably, select a font that is available in most printers, e.g. Arial, Times New Roman or Courier.

The return `code` is the error code of the action; this will be 0 if the action is successful. Any other value signifies the error code of the operating system.

Examples

```
errorcode = setfont ('Times New Roman', 10)
errorcode = setfont ('Arial bold italic', 12)
```

5.2.121 SETLINESPACING

Syntax

```
stat = setlinespacing (value, unit_code)
```

Arguments

```
unit_code: integer
value: numeric
```

Platform

Windows

Meaning

The function `setlinespacing` sets the distance between two lines, including the line itself. (The space occupied by a line of text, including white space until the next line of text.) `stat = 0` means the operation was successful, `stat <> 0` means the operation was unsuccessful.

The following `unit_codes` can be used:

Unit_code	Unit
0	mm
1	cm
2	inch
3	points

Example

```
setlinespacing (2,1).
```

Result

The lines are printed with a distance of two centimeters between them.

5.2.122 SETORIENTATION

Syntax

```
code = setorientation (type)
```

Arguments

code: integer

type: 'portrait' or 'landscape'

Platform

Windows

Meaning

Indicates whether the printing orientation is to be 'portrait' (vertical orientation of paper) or 'landscape' (horizontal orientation of paper). `type` is a string, so it should be embedded in quotes (as shown) when the function is called. The return `code` is the error code of the action; this will be 0 if the action is successful. An error code other than zero signifies the error code of the operating system, and will occur, for instance, if the desired orientation is not supported by the selected printer.

5.2.123 SETPAPERSIZE

Syntax

`code = setpapersize (number)`

Arguments

`code`: integer

`number`: see table below

Platform

Windows

Meaning

Sets the paper format. Each `number` stands for a certain format. The return `code` is the error code of the action; this will be 0 if the action is successful. An error code other than zero signifies the error code of the operating system, and will occur, for instance, if the desired paper size is not supported by the selected printer.

The following formats are supported by `setpapersize`:

Number	Format type
1	Letter 8½ x 11 in
2	Letter Small 8½ x 11 in
3	Tabloid 11 x 17 in
4	Ledger 17 x 11 in
5	Legal 8½ x 14 in
6	Statement 5½ x 8½ in
7	Executive 7¼ x 10½ in
8	A3 297 x 420 mm
9	A4 210 x 297 mm
10	A4 Small 210 x 297 mm

11	A5 148 x 210 mm
12	B4 (JIS) 250 x 354
13	B5 (JIS) 182 x 257 mm
14	Folio 8½ x 13 in
15	Quarto 215 x 275 mm
16	10x14 in
17	11x17 in
18	Note 8½ x 11 in
19	Envelope #9 3 7/8 x 8 7/8 in
20	Envelope #10 4 1/8 x 9½ in
21	Envelope #11 4½ x 10 3/8 in
22	Envelope #12 4 ¾ x 11 in
23	Envelope #14 5 x 11½ in
24	C size sheet
25	D size sheet
26	E size sheet
27	Envelope DL 110 x 220 mm
28	Envelope C5 162 x 229 mm
29	Envelope C3 324 x 458 mm
30	Envelope C4 229 x 324 mm
31	Envelope C6 114 x 162 mm
32	Envelope C65 114 x 229 mm
33	Envelope B4 250 x 353 mm

34	Envelope B5 176 x 250 mm
35	Envelope B6 176 x 125 mm
36	Envelope 110 x 230 mm
37	Envelope Monarch 3 3/8 x 7 1/2 in
38	6 3/4 Envelope 3 5/8 x 6 1/2 in
39	US Std Fanfold 14 7/8 x 11 in
40	German Std Fanfold 8 1/2 x 12 in
41	German Legal Fanfold 8 1/2 x 13 in
The following formats are only supported under Windows 95 and Windows NT 4.0, or higher.	
42	B4 (ISO) 250 x 353 mm
43	Japanese Postcard 100 x 148 mm
44	9 x 11 in
45	10 x 11 in
46	15 x 11 in
47	Envelope Invite 220 x 220 mm
48	RESERVED--DO NOT USE
49	RESERVED--DO NOT USE
50	Letter Extra 9 1/4 x 12 in
51	Legal Extra 9 1/4 x 15 in
52	Tabloid Extra 11.69 x 18 in
53	A4 Extra 9.27 x 12.69 in
54	Letter Transverse 8 1/4 x 11 in

55	A4 Transverse 210 x 297 mm
56	Letter Extra Transverse 9¼ x 12 in
57	SuperA/SuperA/A4 227 x 356 mm
58	SuperB/SuperB/A3 305 x 487 mm
59	Letter Plus 8.5 x 12.69 in
60	A4 Plus 210 x 330 mm
61	A5 Transverse 148 x 210 mm
62	B5 (JIS) Transverse 182 x 257 mm
63	A3 Extra 322 x 445 mm
64	A5 Extra 174 x 235 mm
65	B5 (ISO) Extra 201 x 276 mm
66	A2 420 x 594 mm
67	A3 Transverse 297 x 420 mm
68	A3 Extra Transverse 322 x 445 mm

5.2.124 SETSTATUSBARTEXT

Syntax

```
stat = setstatusbartext (text)
```

Arguments

stat: integer

text: text

Platform

Windows

Meaning

You can use this function to print `text` in the status bar of the window. `stat` will be 0 if the action is successful. Any other value signifies the Adlib error code.

Example

```
stat = setstatusbartext('Now press <Enter>.')
```

Result

If `stat` is 0, the `text` *Now press <Enter>*. will be printed in the status bar of the window.

5.2.125 SETVAR

Syntax

```
code = SETVAR(variable, value)
```

Arguments

code: integer

variable, value: text

Meaning

SETVAR assigns the string `value` to the system variable `variable`. The `code` is 0 if the action was successful. It is also perfectly allowed to empty a system `variable` by assigning an empty `value`, so this counts as a successful action as well; `code` 1 means the action was unsuccessful (with reason unknown).

System (aka environment) variables are not variables defined by the system (although Adlib defines a number of reserved system variables for you implicitly), you define them here with SETVAR in the system, so the variable can have almost any name (without spaces) you choose. However, they are primarily meant for internal use by the Adlib software, to pass different kinds of *adlwin.exe* status information to the *adapl* interpreter. So there are a number of reserved system variables, like the variables starting with an `&` (such as the commonly used `&E` to obtain FACS function result codes to decide what to do next) and the `ADLIB_SMARTHOST` variable which must contain the name of your SMTP server if the *adapl* must be capable of sending e-mails. Normally you don't assign any values to the reserved variables starting with an `&`, because *adlwin.exe* fills them for you, but you must

assign the name of your SMTP server to the `ADLIB_SMARHOST` variable explicitly in the `adapl` from which you want to be able to send e-mails.

Example 1

```
smarthost_result = setvar('ADLIB_SMARHOST', text$(192))
```

Result

This is the typical `ADLIB_SMARHOST` assignment, which can be found in some `adapls`. The name of your SMTP server must be stored in line 192 of the `adlib#.txt` files in the `\texts` subfolder of your Adlib system. However, whether you have done that or not, `smarthost_result` will be zero in either case. So to test for the presence of an SMTP server name, you must test if `text$(192)` is empty or filled.

Example 2

```
smarthost_result = setvar('ADLIB_SMARHOST', '')
```

Result

This assignment empties the `ADLIB_SMARHOST` variable and `smarthost_result` will be zero if it succeeds.

Example 3

```
code = setvar('LAST_ORDER', '17-07-1994-DBX')
```

Result

`code` will be 0 if the value '17-07-1994-DBX' is successfully assigned to the system variable `LAST_ORDER`.

See also

`GETVAR`

5.2.126 SETWINDOWTITLE

Syntax

```
stat = setwindowtitle (title)
```

Arguments

`stat`: integer

`title`: text

Platform

Windows

Meaning

You can use this function to set the `title` of the active window. `stat` is 0 when the action is successful and 1 when it is not.

Example

```
stat = setwindowtitle('Adlib message')
```

Result

If `stat` is 0, the `title` of the active window is set to *Adlib message*.

5.2.127 SHOW

Syntax

```
SHOW(row, col, video foreground ON background, text)
```

Arguments

`row, col`: integer

`video`: see table below

`foreground`: see table below

`background`: see table below

`text`: text

Platform

DOS

Meaning

Displays `text` in accordance with the specified position and attributes. Possibilities include:

Video	Foreground	Background
BLINKING	BLUE	BLUE

BRIGHT	YELLOW	RED
DIM	RED	GREEN
REVERSE	GREEN	CYAN
	CYAN	MAGENTA
	MAGENTA	WHITE
	WHITE	

Example

```
show(1,1,YELLOW ON BLACK,'Y on B, top left-hand corner')
```

Result

At the top left-hand corner of the screen, the text *Y on B, top left-hand corner* will be printed in yellow on a black background.

5.2.128 SKIP

Syntax

```
skip number_lines
```

Arguments

```
number_lines: numeric
```

Meaning

Prints `number_lines` empty lines (to the print file). The command `skip` can also be used in combination with the `output` command. The syntax then is: `output skip number_lines`. The result will be exactly the same in both cases.

Example

```
skip 3
```

Result

Three empty lines are printed (sent to the print file).

See also

OUTPUT

5.2.129 SQRT

Syntax

```
root = SQRT(number)
```

Arguments

```
root, number: numeric
```

Meaning

Returns the square root of `number`.

Example

```
root = sqrt(9)
```

Result

```
root is 3.
```

5.2.130 STATUS

Syntax

```
STATUS
```

Meaning

When compiling a source text, the ADAPL compiler displays the following information:

- The name of the source code file
- The line number in the source code file
- The number of errors and warnings found

For example:

STATUS: Current file test.ada, line = 16, 0 total errors so far.

If the compiler is called with the `'-d'`-option, extended information will be available that can be of use to technical staff at Adlib.

5.2.131 STR\$

Syntax

```
result = STR$(number, format)
```

Arguments

`result`: text

`number`: numeric

`format`: see table below

Meaning

Converts a `number` to a character string according to the indicated `format`.

A series of special characters (conversion characters) indicates how long the character string is to be, as well as the position of the digits, commas, decimal points and signs. The length of the `result` is always equal to the length of the code. The following conversion characters are allowed:

z	Number position	For all occurrences of z a digit will be printed. Leading zeros show as blanks.
#	Number position	For all occurrences of # a digit will be printed. Leading zeros show as zeros.
+	Character position	A single + will cause the sign (+ or -) to be printed in the corresponding position in the output. Multiple + characters will be replaced by printing digits in the number and the sign will be printed in the position immediately to the left of the leftmost printed character. If the format string ends with a +, the sign (- or +) will be in the rightmost position.
-	Character	As +, except that a blank is printed instead of

	position	a + sign if the number is positive.
CR	Character position	The characters CR may only be used in the last two positions (rightmost part) of the string. If the number is negative the letters CR ("credit") will be printed. If positive, two spaces will be printed.
\$	Dollar position	A \$ may only be preceded by a single sign character. A single \$ will cause a \$ to be printed in the corresponding position in the output. Multiple dollar signs will be replaced by printing characters in the number, and a single \$ will be printed in the position immediately to the left of the leftmost printed character.
,	Comma position	The comma can be used as a punctuation mark in the part of a number that precedes the decimal point. The comma is used if the preceding character produces a number, otherwise, a space (or pad character) will be used.
.	Decimal point position	ADAPL uses the full stop as a decimal point. Following the full stop, only fixed numbers (#) and the characters +, - or CR may be used.
*	Pad character	Replaces blanks with asterisks. May be preceded by +, - or \$.

If the number does not fit in the format, the result will be a series of asterisks ('*****') of the same length as the format.

Example

In the interests of clarity, spaces are represented by 'x's.

Number	Format	Result
456	'####'	'0456'
45678	'####'	'****' <i>too large!</i>
0	'####'	'0000'

456	'ZZZZ'	'x456'
4567	'ZZZZ'	'4567'
0	'ZZZZ'	'xxxx'
0	'ZZZ#'	'xxx0'
5.045	'#.##'	'5.05' rounded
0	'#.##'	'0.00'
4567.89	'ZZZ,ZZZ,ZZ#.##'	'xxxxxx4,567.89'
456789.01	'ZZZ,ZZZ,ZZ#.##'	'xxxx456,789.01'
0	'ZZZ,ZZZ,ZZ#.##'	'xxxxxxxxxxx0.00'
4	'+###'	'+004'
-4	'+###'	'-004'
4	'-ZZ#'	'xxx4'
-4	'-ZZ#'	'-xx4'
456	'ZZZZZ+'	'xx456+'
-456	'ZZZZZ+'	'xx456-'
456	'ZZZZZ-'	'xx456x'
-456	'ZZZZZ-'	'xx456-'
45678	'ZZZ,ZZ#CR'	'x45,678xx'
-45678	'ZZZ,ZZ#CR'	'x45,678CR'
456	'+++ ,++#.##'	'xxx+456.00'
-456	'+++ ,++#.##'	'xxx-456.00'
52	'\$ZZZZZZ#'	'\$xxxxx52'
52	'\$\$\$\$\$\$#'	'xxxxx\$52'

456321	'\$***,***,**#.##'	'\$****456,321.00'
--------	--------------------	--------------------

See also

VAL

5.2.132 STRING\$

Syntax

```
result = STRING$(character, number)
```

Arguments

result, character: **text**

number: **integer**

Meaning

Generates a character string of `number` times `character`.

Example

```
result = STRING$('a', 5)
```

Result

result will be 'aaaaa'.

5.2.133 SWITCH

Syntax

```
SWITCH ( expression ) {  
    CASE expression: {  
        code  
        BREAK  
    }  
}
```

```
...
    DEFAULT: ...
}
```

Arguments

`expression` can contain any valid expression. If you provide a mathematical expression for a particular `CASE`, then place that expression between brackets `()`. If you explicitly want to provide a string for a particular `CASE`, then place single quotes around that string (for example when you don't want ADAPL to try to calculate $(x+y)$, but want to have it treated as text.) If you provide the name of a variable without quotes around it, for a particular `CASE`, first the value in it is retrieved. In the end all expressions (calculated sums, retrieved values, and strings) are compared as strings.

In code you can place any ADAPL code you want, except other `SWITCH` statements (nesting can be achieved by putting the other `SWITCH` statements in separate subroutines, to which you can jump from the main `SWITCH CASES`). You can separate commands and functions by a space, or by a semicolon and a space, or by the end of a line. A code line or block is always placed between curly brackets `{}`.

Meaning

A switch/case-construction makes it possible to program nested if/then/else-constructions more elegantly. Especially when a variable can assume many values and you want to execute a different piece of code for each possible value, then `SWITCH` is ideal.

`SWITCH` calculates the value of the first expression and converts it to a string. Subsequently the program jumps to the first `CASE` statement and compares the (string) value of that expression with the first. If they are equal, the code line or block following that first `CASE` is executed. If not, then the next `CASE` will be tested, and so on.

Just like the entire `SWITCH` statement, the code is being run through from top to bottom, including the `DEFAULT`, and the `BREAK` command ensures that the `SWITCH` structure is not being run through any further, and execution jumps to the line following the last `SWITCH` curly bracket. (`BREAK` is also needed in the last `CASE` block if you don't want the `DEFAULT` to be executed if this `CASE` is true.) If you don't use `BREAK`, every next `CASE` is being tested and/or run through until finally `DEFAULT` is being executed. (For `DEFAULT` you have to provide some

code. If you don't really need any default code, just use `DEFAULT: BREAK`, to end the `SWITCH` construction.) The `DEFAULT` will always be executed if not a single `CASE` test is positive, and this is what `DEFAULT` is really meant for. The `DEFAULT` is placed below the `CASE` statements.

The opening curly bracket must always be placed on the same line as the `SWITCH`, `CASE` or `DEFAULT` statement. The closing curly bracket can be on a line of its own.

Note that in between the lines that hold the `SWITCH` and `CASE` statements, you cannot place an empty line or any comments: placing an empty line or comment line here will make it impossible to compile this adapl.

Example

```
switch (myvar) {
  case '2': { display 'myvar is'; display 'two'; break;}
  case (2 + 1): {
    display 'three'
    break
  }
  case 4: {display 'four' break}
  default: display 'myvar is greater than four'
}
```

Result

When `myvar` contains anything different from `'2'`, `'3'` or `'4'` than *myvar is greater than four* is being displayed.

When `myvar` contains `'2'` than *myvar is two* is being displayed, and ADAPL leaves the switch construction.

When `myvar` contains `'3'` than *three* is being displayed, and ADAPL leaves the switch construction.

When `myvar` contains `'4'` than *four* is being displayed, and ADAPL leaves the switch construction.

5.2.134 SYSTEM

Syntax

```
stat = SYSTEM('<executable_program>')
```

Arguments

stat: integer

executable_program: text

Meaning

The text `executable_program` is passed as command line to the operating system, e.g. to start an external program and then go back to the ADAPL program. If the command is successfully carried out, the result in `stat` will be 0. Any other value signifies an error.

You can use `system` also to execute internal system commands, to manipulate files and directories like you would under DOS. You can always do this by calling `cmd.exe` (part of your operating system), and pass the internal command and options as arguments in the following syntax:

```
stat = SYSTEM('cmd /c <intcmd> <file_or_dir>')
```

(Parameters between `<` and `>` must be replaced by an actual value. Use single quotes.) If a file name contains spaces, the entire file name must be enclosed by double quotes. You can use local, relative or UNC paths. For a complete description of this `cmd.exe` function, type `help` in a (DOS) command window, or type `help <function name>`. The problem with `cmd.exe` though, is that `stat` won't contain the result code of the internal command (which you really want to know) but the result code of `cmd.exe` instead. So when you were to rename a non-existing file this way, resulting in a system error, then `stat` would still contain 0 because `cmd.exe` was executed alright.

Some internal system commands that you can use without calling `cmd.exe` are:

ATTRIB

CHDIR

COPY

DEL (ERASE)

MKDIR (MD)

RENAME (REN)

RMDIR (RD)

CD (change directory)

For these selected commands, the syntax becomes:

```
stat = SYSTEM('<intcmd> <file_or_dir>')
```

The advantage being that `stat` will result in the error code of the internal command, as desired.

Note that you cannot use Adlib variables or tags as arguments for the internal command, but you *can* use an Adlib variable as the argument for `SYSTEM`. So if you want to use variable file names (maybe coming from an Adlib field), you'll have to build up the entire `SYSTEM` argument string in a variable and pass the variable to `SYSTEM`, like so for example:

```
argumentstringvar = 'rename ' + pathplusfilename1var + ~  
                    ' ' + filename2var  
resultvar = system(argumentstringvar)
```

NB: With the `SYSTEM` function, a program can be started that disrupts execution of Adlib or ADAPL - with unpredictable results.

Also note that this command has partly been replaced by the `LAUNCH` function

Example 1

```
stat = SYSTEM('adlwin')
```

Result

`stat` gets the value 0 if `adlwin.exe` has successfully been started from the current folder, and otherwise an error code from the operating system.

Example 2

```
stat = SYSTEM('del log.txt')
```

Result

`stat` gets the value 0 if the file `log.txt` has been removed successfully, and otherwise an error code from the operating system.

5.2.135 TAG2FIELD\$

Syntax

```
fieldname = tag2field$ (tag)
```

Arguments

fieldname: text

tag: database variable

Meaning

If the command has been successfully executed, `fieldname` will contain the fieldname of the `tag`. If no `fieldname` has been provided for this tag in the database, `fieldname` will get the `tag` itself as value. If an error occurs, `fieldname` will be an empty string. If you invoke this function from a stand-alone `adapl`, you will get the name as it was filled in for the default language. `Adlib` first checks if the name is available in the active language, if not, it will look for the default language.

Example

```
fieldname = TAG2FIELD$ ('ti')
```

Result

`fieldname` will contain the name that the tag `ti` has in the active database. This will probably be *Title*.

5.2.136 TEXT\$

Syntax

```
message = TEXT$ (handle, textnumber)
```

Arguments

message: text

handle, textnumber: integer

Meaning

`TEXT$` fills `message` with the text from line number `textnumber` of the

text file referred to with `handle`.

The text file referred to with `handle` must have been opened with `openfile` first.

It is also possible to put all texts used by `adapls` in one text file (per language), and open the right language variant in the running application automatically, whenever an `adapl` needs to read from a text file, by just providing the name of this `.txt` file once, in the *Adapl* option in the application setup.

The advantage of this option is that you no longer have to open specific text files in `adapls` before you can read from them. You can just always read from the text file specified in the application setup (without providing a `handle`). You then only have to provide the `textnumber`. The syntax becomes:

```
message = TEXT$ (textnumber)
```

See also

Handling text files

5.2.137 TIME\$

Syntax

```
time = TIME$(1)
```

Arguments

```
time: text
```

Meaning

Returns the system `time` as a character string in the form *hh:mm:ss*. The argument currently has no significance, but has to be 1.

5.2.138 TITLE

Syntax

```
TITLE 'text'
```

Meaning

Assigns 'text' to the ADAPL program as title. If the ADAPL is included in a list of available programs, the title is displayed alongside it.

The single quotation marks are mandatory.

In the case of multilingual applications, `title` stands for the title in the default language and `title[X]=` stands for the title in language X, with the X denoting the number of the language in Adlib.

5.2.139 TOSTRING\$

Syntax

```
display_value = tostring$ (tag[occ])
```

Arguments

`display_value`: text

`tag`: database variable

`occ`: integer

Meaning

The format of a value stored in a numeric or date field may differ from the way it is presented in the application. For example, an ISO date field may have the presentation format *European date*. Likewise, a numeric field is usually stored with a dot as decimal separator, but may be presented on screen with a comma (if set up this way).

The `tostring$` function allows ADAPL to reformat the stored numeric or date value to the presentation format set up for the field. Numbers and dates will then be converted to a string. This way, you can print these values in the format in which they are shown on screen.

Example

Assume a date field with tag DS has the data type *ISO date (yyyy-mm-dd)* and the presentation format *European date (dd/mm/yyyy)*. The stored value (in occurrence 1) is 2010-07-06. If `display_value` is declared as a text variable, ADAPL converts this date to the European format as follows:

```
display_value = tostring$ (DS)
```

Result

displayvalue will be: 06/07/2010

5.2.140 TRANSFORM

syntax

```
result = TRANSFORM (FACS database, stylesheet, file name)
```

Arguments

FACS database: FACS database name

result, stylesheet, file name: text

Meaning

With the `TRANSFORM` function you can transform an Adlib record by means of your own XSLT stylesheet, from the Adlib XML format to every other desired document format. As arguments to the function, you include a FACS database, the name of the stylesheet and optionally a (relative path plus) file name into which the `result` is saved as a file. (If you provide a file name, also add the proper extension of the format into which you are transforming the record, e.g. `.htm` or `.txt`.) The record to be transformed must already have been read (via `READ`) from the relevant, opened FACS database. This way, you can transform a record to plain text, for example, or to an HTML page, to be sent by e-mail or for display in an internet browser.

A FACS database name is an alias for an Adlib database, which is defined in the current `adapl`. To be able to use a record from this database in this function, that database must have been opened earlier in the `adapl` (with the `OPEN` function) and the desired record must have been read (with the `READ` function). The FACS database name must be entered without quotes.

Via your own XSLT stylesheet you specify the transformation that has to be performed. Adlib XML is structured according to the `adlibXML.xsd` (an XML Schema Definition). The most important thing you need to know about this is that only the XML tags of the three highest levels have been specified, namely: `adlibXML` (root tag), `recordlist` (may occur only once), `record` (may occur indefinitely). Further, there is a `diagnostic` tag on the level of the `recordlist`, which contains metadata. The structure of an Adlib record itself is not defined in the schema definition because this differs per database. It's easy though, to obtain an example of Adlib XML, by exporting a few marked records to an XML file from your Adlib application.

In this function, you enter the stylesheet via its full file name, including the extension (`.xsl` or `.xslt`) enclosed by single quotes.

If you provide a file name (which is optional), enclose it by single quotes.

Example 1

```
result = transform (COLLECT, 'AdlibXML-toHTML.xsl')
sendmail ('me@adlibsoft.com', 'you@museum.com', '', 'new
records', result, '', 1)
```

Result

First, the record currently read from FACS database COLLECT is transformed from XML to HTML via a stylesheet created for this purpose especially. This HTML page is stored temporarily in the `result` variable. Then, an e-mail is sent, with the contents of `result` as message body text. The argument 1 indicates that the body text must be interpreted as HTML, not as plain text.

5.2.141 TRIM\$

Syntax

```
new_string = TRIM$(text, code)
```

Arguments

`text`, `new_string`: text

`code`: integer

Meaning

`trim$` can be used to remove leading and trailing blanks from a character string. The codes for this are:

Code	Meaning
1	Leading blanks are removed
2	Trailing blanks are removed
3	Leading and trailing blanks are removed

Example

```
new_string = trim$(' ABC ', 3)
```

Result

```
new_string is 'ABC'.
```

5.2.142 UNLOCK

Syntax

```
UNLOCK facsname
```

Meaning

Use `UNLOCK` to remove the write lock from a record if this is not done by `WRITE` or `DELETE`.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.143 USER\$

Syntax

```
result = USER$(code)
```

Arguments

```
result: text
```

```
code: integer
```

Meaning

Returns the user name (as `code = 1`) or the user number (as `code = 2`) or the 'friendly' user name (as `code = 3`). Both versions of the user name are extracted from Active Directory (namely from the *Logon name*, respectively the *Display name*) and pertain to the user currently logged in to Adlib, or to Windows (the latter if no login procedure has been set for Adlib). Of course, the 'friendly' version of the user name

can only be retrieved if it has been registered in Active Directory by the system administrator. The `result` is always a text string. On a single-user system, the user number will always be '0', and the username ' ' (the null string).

Example 1

```
name = user$(1)
```

Result

```
name is: MIKE.
```

Example 2

```
number = user$(2)
```

Result

```
number is: 8.
```

5.2.144 VAL

Syntax

```
number = VAL(text)
```

Arguments

```
number: numeric
```

```
text: text
```

Meaning

Returns the numeric value of a character string. You must always use this function when you want to calculate something using values from tags (whether those tags are FACS tags or taken directly from the currently open record), and when you want to numerically compare values, because in adapls, content from field tags is always of the text type.

Also note that sums like: `tag1 = (val(tag2) + val(tag3))` really need the outer brackets around the whole addition to assign the numerical sum to `tag1`, otherwise `val(tag2)` will be assigned to `tag1` first, turning it into a string, and then `val(tag3)` will be added to `tag1`, making it a string addition.

Example

```
text = '1234.5'  
number = VAL(text)
```

Result

`number` will be *1234.5* and can be used in calculations. `text` (a string which, in this situation, has 6 characters) cannot be used in calculations.

See also

STR\$

5.2.145 WAIT ... NOWAIT

Syntax

```
wait  
instruction  
instruction  
instruction  
...  
nowait
```

Meaning

These functions switch the hourglass cursor on and off, respectively. This is useful for time-consuming operations in ADAPL.

5.2.146 WHATDATE

Syntax

```
new_date = WHATDATE(date, days)
```

Arguments

```
new_date, date: text
```

```
days: integer
```

Meaning

The argument `date` must be in Julian (yy.ddd), European (dd/mm/yy) or ISO date format (yy-mm-dd). `new_date` is calculated by adding the number of days in the parameter `days` to the date in the parameter `date`. If `days` is negative, `new_date` will precede `date`. The `new_date` calculated must come after 1 January 1900. For `date` you can also use 'today' or `date$(<code>)`, representing the system date. The system will then automatically fill in the value that is valid at the time the adapt is carried out.

The output date format will be the same as the date format you put in.

Example 1

```
new_date = whatdate('22/07/88', 100)
```

Result

```
new_date is: '30/10/88'
```

Example 2

```
new_date = whatdate('30/10/1988', -100)
```

Result

```
new_date is: '22/07/1988'
```

Example 3

```
new_date = whatdate(date$(8), -100)
```

Result (if today's date is 1988-10-30)

```
new_date is: '1988-07-22'
```

Example 4

```
new_date = whatdate('today', 1)
```

Result (if today is 20/05/02)

`new_date` is: '21/05/02'

See also

`DATE$`

5.2.147 WHILE

Syntax

```
WHILE expression instruction
```

or:

```
WHILE expression {
```

```
instruction
```

```
instruction
```

```
...
```

```
}
```

Meaning

The `instruction` or `instruction` block is only carried out if the `expression` results in `TRUE`. The entire block is then repeated for as long as the `expression` remains `TRUE`.

See also

Conditional loops

5.2.148 WORDCREATEDOCUMENT

syntax

```
result = WordCreateDocument (template, document_or_printer,  
option)
```

Arguments

template, document_or_printer: text

option: numeric

result: integer (error code)

Meaning

With Microsoft Word (version 97 or higher), you can use ADAPL to place data from one or more Adlib records into a Word template to automatically make one or more documents. You can also print directly, if desired.

Because the data is exported to a Word document that can be edited, `WordCreateDocument` allows you the freedom to use MS Word to change the format and layout of your exported data, before you actually print the document(s) from Word (or afterwards).

Use the arguments as follows:

- **template** is a text variable or constant with the name of the (custom) Word template that you want to use.
- **document_or_printer** is a text variable or constant that may contain the file name in which the resulting document is to be stored (a Word document). So the file name is used to create the document and to save it when it's completed. If you use a relative path in front of the file name, make sure it is relative to the application folder. Also, folder names used in paths (absolute or relative) must already exist, otherwise no documents will be generated.

For the `option` values 50, 112 and 340 (see below), you must provide the name of the printer to which you want to print directly (e.g. 'HP LaserJet 5Si MX'), instead of providing a document name. Or you may leave this option empty, to print to the default printer. But when printing directly, the created document will not be saved automatically, because it has no name.

- **option** is a numerical variable or constant that determines a few settings (see table below).
- **result** is 0 when the operation was successful, otherwise an error code results.

Option	Meaning
0	MS Word will not be visible. With this option value, you can (also) export multiple records, although each record

	will be inserted in its own document. If you do want to export more than one record with this option, you should use a text variable in which you build up a new document name for each record, for instance by adding some incremented number to a fixed name. Otherwise only the last record will be saved in the specified document.
1	MS Word will be visible and appears in a separate window on top of your Adlib application. You can export more than one record with this option, although each record will get its own document, and still specify only one document name in the function arguments. The result will be that all record documents will be opened, and that only the first will automatically receive the specified name; the rest is nameless. You'll have to name and save all these documents manually (if desired). If you just want to print the opened documents, you don't need to name and save them.
16	With <code>WordCreateDocument</code> it is also possible to export all selected records to one Word document (which may result in a real performance boost). For this purpose the three (compound) values 16, 18, and 20 are available. Use the value 16 for all selected records that are not the first or the last record. Those records will then be exported to the same document as the first, already exported, record.
18	To export multiple selected records to one document, use the three (compound) values 16, 18, and 20. For creating a new document and exporting the first selected record, you must use 18.
20	To export multiple selected records to one document, use the three (compound) values 16, 18, and 20. Use 20 for the last record, to finish building up the document after exporting the last record.
50	A disadvantage of <code>WordCreateDocument</code> is sometimes that the created documents remain opened in Word and are not printed automatically. You may of course use the <code>WordPrintDocument</code> function for this purpose, but that has the disadvantage that every record will be printed in its own document (page), and that Word reports each individual successful printout (which may be annoying when printing many reminders, for instance). However, it turned out to be difficult (because of the necessity for backward compatibility) to extend <code>WordPrintDocument</code> with options to print all records from one document, and

	that's why another three extra (compound) values have been made available for the <code>option</code> in <code>WordCreateDocument</code> , namely 50, 112 and 340, to allow direct (automatic) printing with this function too. For creating a new document, exporting the first selected record, and preparing the document to be built up for direct printing, you must use 50.
112	Use the value 112 for all selected records that are not the first or the last record, and for preparing the document to be built up for direct printing. Those records will then be exported to the same document as the first, already exported, record.
340	Use 340 for the last record, to finish building up the document after exporting the last record, and have it printed directly.
	Note that the template which is used for printing, will be visible in Word. The document that is being built, won't be visible when you use 16, 18, 20, 50, 112 or 340. If you do want it to be visible, you'll have to add 1 to all values.

In your print adapt, the value of the `option` has to be determined conditionally if you wish to export multiple records to one document. This means that the adapt has to figure out itself whether the current record is the first, the last, or one in between. And dependent on the outcome, the `option` must be filled with an appropriate value to control the printing process.

Example 1 (for Adlib Library)

```
WordCreateDocument ('c:\bibl\wincat\mytemplate.dot', 'c:\test.doc', option)
```

Example 2 (for Adlib Museum Standard)

```
WordCreateDocument ('c:\adlib\standard\mytemplate.dot', docnamevariable, 0)
```

All option values explained (for advanced users)

The range of compound values that the `option` of `WordCreateDocument` can assume, is determined in principle by the possible add-up combinations of all basic values that represent individual functionality. The basic values are numbers that cannot be formed by adding smaller basic values to each other, and therefore they become

exponentially larger. For example: in several ADAPL functions, an option can have the values 0, 1, 2 and 4. If ADAPL interprets the value in such an option, it must be able to distinguish a basic value from an add-up combination of other basic values (and therefore a combination of the functionality that each basic value represents). For this reason, the value 3 can never be a basic value because 3 can be formed by adding 1 to 2. Now it is clear that any new option value for extra functionality in this example, must receive the value 8, because 5, 6 and 7 can be formed by add-up combinations of 1, 2 and 4. The basic values of the `option` in `WordCreateDocument` are now as follows:

Basic values	Functionality
0	The Word document that is being built up, won't be visible.
1	The Word document that is being built up, will be visible.
2	Start making a document and export the current record. (In this function this only works if you add 16.)
4	Export the current record and finish building up the document. (In this function this only works if you add 16.)
8	(Functionality that isn't applicable to this function.)
16	Export the current record to the same document as the previous or next record.
32	For direct printing of a document to be built up: keeps the current template in memory, for following records. Use this for all records, except the last one.
64	For direct printing of a document to be built up: for the current record the same template will be used as for the previous record. (This may appear a redundant option considering 16, but is still necessary anyway.)
128	For direct printing of a document to be built up (also use 256): does not display the template that is being used. In 5.0.3 however, this option is not operational, so you can't hide the template to be used! For compound values it doesn't matter if you include 128 or not.
256	Build up the document for direct printing, and do not save the document. In principle, this value needs only to be used when exporting the last record. But if problems arise, 256 may also be used for the other

	records.
512	Do not export the current record, and finish building up the document.

The basic values in this list can in principle be added up to obtain their combined functionality. However, most combinations do not make any sense or cause problems. The compound values advised in the previous paragraph are built up as follows:1846

$$16=16+0$$

$$18=16+2$$

$$20=16+4$$

$$50=2+16+32$$

$$112=16+32+64$$

$$340=4+16+64+256$$

To the last three compound values you may add 1, if necessary, and to the first two of those last three possibly (also) 256.

See also

Accessing data for a Word template from an adapl

5.2.149 WORDCREATELABELS

syntax

```
result = WordCreateLabels (labels_template, document,
option, occurrence)
```

Arguments

labels_template, document: **text**

option, occurrence: **numeric**

result: **integer (error code)**

Meaning

With Microsoft Word (version 97 or higher), you can use ADAPL to place data from Adlib into a Word labels template to automatically make a labels document.

WordCreateLabels allows you the freedom to use MS Word to change the format and layout of your data. For example, you may want to produce address labels using Adlib data.

Use the arguments as follows:

- **labels_template** is a text variable or constant with the name of the (custom) Word labels template that you want to use.
- **document** is a text variable or constant that includes the file name in which the resulting document is to be stored (a Word document).
- **option** is a numerical variable or constant that determines a few settings (see table below).
- **occurrence** is a numerical value or constant, which has to be zero if you want to print all occurrences of the current record, or the number of a specific occurrence if you only want to print that occurrence.
- **result** is 0 when the operation was succesful, otherwise an error code results.

Option	Meaning
1	MS Word will be visible and appears in a separate window on top of your Adlib application.
2	Start making a document and use the current record.
4	Use the current record and finish the document.
8	Print all occurrences of the current record in one label. This may be handy to print data of all copies of a book on one label, for example.
	<p>Note that you can add these options if desired, to obtain their combined functionality.</p> <p>If you enter 6 (2 + 4), you start a document, print the current record, and end making the document. MS Word won't be visible during printing, and for each occurrence of the current record a label will be printed in the current document. If you use option 6 for all records, then for each record in the selection a new document will be created.</p> <p>If you'd like all records in the selection to be printed together in one document, you will have to use option 2 (or 3) for the first record in the selection only, option 1 (or 0) for all the following records, and 4 (or 5) for the last. (You can also add 8 to these values if necessary.)</p> <p>Note that you always have to start and close a document being made explicitly! (The document will only be created</p>

	after it has been built up in memory and has been finished explicitly.)
--	---

Example 1 (for Adlib Library)

```
WordCreateLabels ('c:\bibl\wincat\mytemplate.dot', 'c:\test.doc', 6, 0)
```

Example 2 (for Adlib Museum Standard)

```
WordCreateLabels ('c:\adlib\standard\mytemplate.dot', 'c:\test.doc', MyOption, 1)
```

Here, `MyOption` is a custom variable that may hold different values dependent on program flow that checks whether the current record is the first in the selection, one in the middle or the last.

See also

Accessing data for a Word template from an adapt

5.2.150 WORDPRINTDOCUMENT

syntax

```
result = WordPrintDocument (template, printer, number)
```

Arguments

template, printer: text

number: numeric

result: integer (error code)

Meaning

Instead of `WordCreateDocument` (that only exports records to a document, and leaves it to you to actually print the opened document) you can also print or e-mail data from a record directly, using a Word template. For this you use the `WordPrintDocument` function. Each exported record will be printed on a separate page.

Use the arguments as follows:

- **template** is a text variable or constant with the name of the (custom) Word template that you want to use.
- **printer** is a text variable or constant with the name of the printer to be used for printing. This includes the name of the server on which the printer is installed, twice; see the example below. Use an empty string (two single quotes without a space between them), to direct output to the default printer.
- **number** is a numerical variable or constant used to determine the number of prints you make. If `number` is 0, zero copies will be printed. You can use the option of printing zero copies, to e-mail the generated Word document instead (as an attachment). In that case, the used Word template must contain relevant e-mail parameters - see the general Adlib User Guide for an explanation of these parameters.
- **result** is 0 when the operation was successful, otherwise an error code results.

Example (for Adlib Library)

```
WordPrintDocument ('c:\bibl\wincat\mytemplate', '\\saturnus
\HP LaserJet 5Si MX on \\saturnus', 1)
```

See also

Accessing data for a Word template from an adap1

5.2.151 WRITE

Syntax

```
WRITE facsname
WRITE facsname UPDATE
WRITE facsname NEXT
WRITE facsname NEXT UPDATE
```

Meaning

The `WRITE` instruction writes a record to the dataset referred to by `facsname`. The current values of the database variables declared for the dataset are written to the record on disk. Non-declared tags remain unchanged. The file referred to by `facsname` must first be opened.

With the `UPDATE` option, the found record is locked while it is being written. Another way to lock the record is to use `LOCK facsname` before writing or deleting. It's important though to lock a record prior to each type of `WRITE` operation, `WRITEEMPTY` or `DELETE`, and unlock it afterwards via `LOCK facsname` and `UNLOCK facsname` respectively. This prevents a record being kept locked unintentionally after editing, which would make it impossible for others to edit it again.

So for a `WRITE` without the `NEXT` option, a record must first have been read with `READ...UPDATE`, or the record must be locked with `LOCK facsname`.

`UPDATE` prevents the lock from being removed from the record. Without this option, the lock will automatically be removed after a successful `WRITE`. A `WRITE NEXT` will not place a lock on the record just created, unless the `UPDATE` option is used.

After a successful `WRITE`, the status variable `&E` will be set to 0.

`WRITE...NEXT` creates a new record in the dataset, containing the current values of the database variables.

See also

[Click here for more information about how to define a FACS name in the application setup.](#)

[Click here for general information about FACS.](#)

5.2.152 WRITEEMPTY

Syntax

```
WRITEEMPTY (facsname)
```

Meaning

The `WRITEEMPTY` instruction creates an empty record in the file referred to by `facsname`. The purpose of this is to obtain the future record number in a before storage ADAPL. Otherwise you would not know the record number until the record was written.

It's important to lock a record prior to each type of `WRITE` operation, `WRITEEMPTY` or `DELETE`, and unlock it afterwards via `LOCK facsname` and `UNLOCK facsname` respectively. This prevents a record being kept locked unintentionally after editing, which would make it impossible

for others to edit it again.

Example

```
if (val (%0) = 0) {
    /* only if new record
    writeempty (_LOCAL)
}
```

Result

The record is stored and %0, the record number tag, is filled. This is only relevant in the case of a new record, i.e. if the record number is still '0'.

NB: `_LOCAL` is a special facsname for general use. `_LOCAL` points simply to the currently opened file. So for reading and writing of records it's not necessary to hard code the facsname of the file.

See also

[Click here](#) for more information about how to define a FACS name in the application setup.

[Click here](#) for general information about FACS.

5.2.153 YESNO

Syntax

```
answer = yesno (window_title, message, yestxt, notxt)
```

Arguments

answer: integer

window_title, message, yestxt, notxt: text

Platform

Windows

Meaning

The user is asked to click either `yestxt` or `notxt` in a window. If the user clicks `yestxt`, `answer` will be 1, otherwise 0.

6 Managing translations

§

6.1 The Translations manager: introduction

The *Translations manager* is for viewing, editing and/or translating all interface texts in your Adlib system in one overview (except for the Help files, at the moment). The data in your database cannot be accessed this way. The files that can be opened in this tool are application structure files (*.pbk*), database structures (*.inf*), screen files (*.fmt*) and system text files (*.txt*). From these files, the texts you can edit and/or translate are message texts, button and screen field/box labels, enumeration texts, method (access point) texts, data source list texts, screen titles, application titles, and field names. All these texts can be edited or translated into as many languages as you wish to make available to users. If text files are in Unicode, you can make translations to all languages supported by Unicode. Editing these texts for existing applications and databases is no problem: the functioning of your Adlib system is not dependent on these texts*. Your only real consideration should be that the length of screen field labels is often limited by the screen layout; this means you need to check (for instance in your running application) if new label texts still fit in the space that is available for it. Also note that if you display Adlib with a larger font, fewer characters will fit in that available space. (If compromising is not satisfactory, you can always edit the screen layout itself of course, in the *Screen editor*.)

* In general, you can change field names in your existing databases without getting into trouble, because although many properties in Designer can hold tags as well as field names, it is always the tag that is stored. The only location where field names may be stored instead of tags, is in Word templates. So if you change field names, check your Word templates for old field names that need to be changed too.

The texts you edit here, can of course also be edited in the files from which they have been extracted, in the *Screen editor*, the *Application browser* or text editors. And naturally, the changes you save here, will be reflected in those files when you open them in their proprietary tools. The advantages of editing texts in one overview in the *Translations manager* are that you no longer have to find out in which file certain texts to be translated are located, and that you no longer

have to do extensive navigating through nested options to edit just a few lines of text. In the overview you can easily find any text in any file, all at once: just sort on a column of your choice and scroll to the texts you are searching for.

See also

Accessing the Translations manager

Editing or translating Adlib interface texts

Saving modifications

6.2 Interface functionality for translations

§

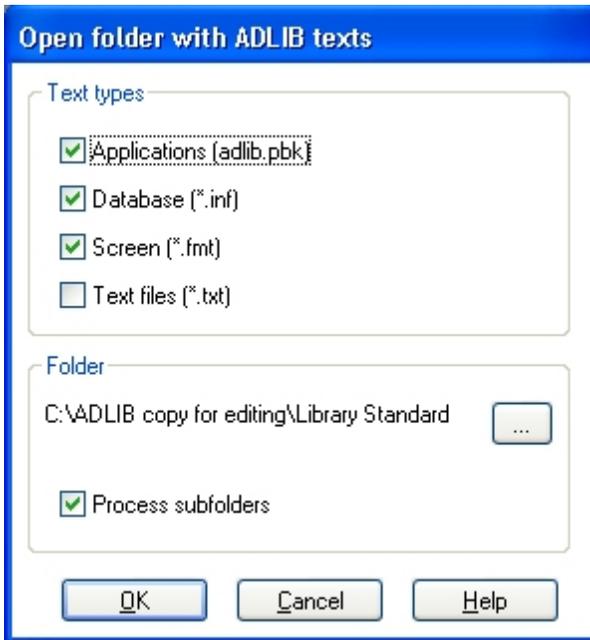
6.2.1 Accessing the Translations manager

To access the *Translations manager* in Adlib Designer, follow these steps:

1. In the main *Adlib Designer* window that opens when you start Designer, start the *Translations manager* by choosing *View > Translations manager* or clicking the button for this tool:



The *Translations manager* opens with the following window in front of it:



2. In the *Open folder with Adlib texts* window, choose a work folder and the file types in which you want to edit or translate texts. As your work folder, choose the main (copy of an) Adlib folder on your system if you want to access texts in your whole application, by clicking the ... button; then select the *Process subfolders* option to access files in subfolders of this work folder too. Or choose a specific subfolder of the main Adlib folder as your work folder, if you want to access only a certain part of all texts. Also select up to four *Text types* that you want to open in the *Translations manager*. You'll have to select them all if you want to translate all aspects of the chosen Adlib application and its databases (except for the Help files and the data content itself). What Adlib files are located in which folders depends on your application. See the Help topic: Adlib file types and folders, for more information.
3. Click OK to open the desired folder and texts in the *Translations manager*, or click *Cancel* to close this window and access the *Translations manager* without setting a (new) work folder and files types. If you opened texts in the *Translations manager* earlier in this session with Adlib Designer, its content will remain unchanged if you clicked *Cancel*; if the *Translations manager* is still empty, or if you want to open other texts, you can call up the *Open folder with Adlib texts* window again, by clicking the *Open*

work folder button:



You can also *Add* the texts of other files or an application folder to the currently opened texts, with the appropriate options in the *File* menu. From there you can also clear the overview.

4. In table view of the *Translations manager* you may directly start editing or translating all texts in the language columns.

See also

The Translations manager: introduction

Editing or translating Adlib interface texts

Saving modifications

6.2.2 Editing and translating interface texts

In the *Translations manager* you'll see an *Adlib objects text table* (the overview), and a *Statistics* tab on the right that indicates the number of opened files and texts, and the amount of translations that are still missing for every language.

The *Adlib objects text table* has the following columns:

- The first column displays a temporary number that a single piece of text and its translation has been given in this overview. The default sorting that it implies is based on the *Source file* column first and secondly on the *Tag* column.
- The second and third column display two translations of the same text. You can select the languages for display in these columns in the *Language 1* and *Language 2* menu's.
- The *Source file* column displays the file name from which the texts in that line have been extracted.
- The *Text type* column explains the type of texts this line holds, so for instance if it is a screen field label or an enumeration text, or a system text, etc.
- The last column (*Tag*) displays either the line number of a system text, an Adlib tag if the text belongs to a field, or a name or a hyphen if the item has no number or tag.

You can sort on each of these columns by clicking the desired column header once; click again if you want to reverse the sorting order of that column.

Editing and/or translating a single text

Just click any text in one of the language columns and type or delete text in it.

Click the *Undo last edit* button for each previous edit you want restore:



From the *Tools* menu, you can start three subtools: *Copy language column*, *AutoTranslate* and *Set initial capitals* to ease repetitive translations. The *AutoTranslate* tool can also be started from the toolbar:



And just like in other Windows software you can cut, copy and paste selected texts (only single texts): see the options in the *Edit* menu.

Find or search-and-replace words

You can search for certain words or partial words in the overview. Open the *Find and Replace* window on the *Find* tab, either by choosing *Edit > Find*, by pressing **Ctrl+F**, or by clicking the *Search in texts* button:



Type any term or part thereof, that you want to search in both displayed language columns or in only one of them.

If the term you type is only part of a word you look for (or can be part of a word), then unmark the *Match words* option. If you mark this option, the search term must appear as a whole word in a text consisting of one or more words.

Mark the *Ignore case* option if upper and lower case are not important while searching.

Then also select the options for the *Language* columns in which you want to search.

Click *Find next* to start the search. A found term is indicated by a short black arrow to the left of a text number. To search a next appearance of the searched term, each time click the button again or press **F3**.

On the *Replace* tab you may extend your search to replacing any found words or parts thereof. Simply enter the word or part of a word to replace a found term with, in the *Replace with* entry field.

Click *Find next* repeatedly until you find an occurrence of the searched term you want to replace, and then click the *Replace* button. Repeat this procedure until all texts have been searched.

You may also skip finding next occurrences and click *Replace all* directly from the start: this finds and replaces all occurrences of the searched term automatically. You cannot confirm replacements, so only use this function if you are absolutely sure that the searched word appears only in texts which you want to change.

Search and replace texts semi-automatically

Start the *AutoTranslate* tool by choosing *Tools > Auto Translate* in the *Translations manager*.

For the currently selected text in the *Adlib objects text table*, *AutoTranslate* shows the translations in both selected languages in the upper part of the window above the list. Both are considered translations.

In the *Translation* list you see all texts in one of the two languages, that have the selected text in them, e.g. "title" appears also in "Source title" and "By title". You also see how many times a text occurs. You can switch the language of this list by clicking the button with no text on it.

In the bottom entry field you type the translation in the language selected for the *Translation* list, with which you want to replace the selected text (always the first one in the list); click the *Translate* button to replace only this occurrence of this text. Click *Translate all* to replace all occurrences of this text.

You can also double-click one of the texts in the list to copy it to the entry field.

Copy the texts of an entire language column

Use the *Copy language column* tool to copy all texts from a language column to another language column. When adding a translation in a new language it may ease translating if you first copy the texts from another column into the new one, for instance because in the new translation many texts will remain in English; then you don't have to enter these terms all over again manually.

Start this tool by choosing *Tools > Copy columns* in the *Translations manager*. In the tool window, select from and to which language you want to do the copying (both languages have to be visible in the *Adlib objects text table* already), and if you do not want the copy process to overwrite texts already present in the "copy to" column, then also mark the *Copy only if destination is empty* checkbox. (The *Copy selected rows only* option is not functional yet.) Click *OK* to start the copying.

Convert the first character of selected text types to capitals

The *Set initial capitals* tool is a simple way to automatically ensure that all texts of specific types start with a capital.

Start this tool by choosing *Tools > Initial capitals* in the *Translations manager*. In the tool window, just select all desired text types to apply this conversion to, and under *Language columns to modify*, select one specific language, or all languages at once, to which the changes should apply, and then click *OK*.

Note that this conversion is executed once when you use this tool, but after the conversion it's up to you again: you can enter new texts or edit converted texts, and have them started without initial capital.

See also

The Translations manager: introduction

Accessing the Translations manager

Saving modifications

6.2.3 Saving your work

At any moment you can save all the texts that you edited or translated, into the files from which those texts were extracted before display in the *Adlib objects text table*.

Do note that the *Translations manager* only notices a change in a text cell, after you have left that cell. This means that after you've edited the last cell, you must first click some other cell to inform the *Translations manager* of that last change.

Open the *Save objects* window by choosing *Save* in the *File* menu or by clicking the *Save all modified texts* button:



In the *Save objects* window you will be presented with an overview of all the unsaved files in which you made changes, and you can determine of each of these separately whether you want to save them or not, by selecting them or deselecting them. Click *Yes* to save the selected files. *Cancel* returns you to the *Translations manager*.

When you close Adlib Designer while you haven't saved all changes yet, the *Save objects* window will automatically appear, allowing you to save your work before the application is closed. Choose *No* to exit Designer without saving.

See also

The Translations manager: introduction

Accessing the Translations manager

Editing or translating Adlib interface texts

7 Import and export

§

7.1 Exchanging data: introduction

It is unlikely that you or your co-workers will always input all the information in the Adlib databases manually. Especially when you are starting out with Adlib, you can save a great deal of time by using any existing databases that were created before in other software and/or by another organization. This particularly applies to catalogue information. In such cases you would like to read in those existing databases into your Adlib system. This is called importing data.

Easier said than done, however. There are numerous ways to organize and code a database, and virtually every database program has its own method of doing so. Such a method is known as a database format. In addition, each database has its own layout for fields. The number of possible field layouts is enormous, particularly because every user of a database program can decide how he or she wants the layout.

Because of this enormous variety in database formats, you must often use a so-called exchange file to be able to exchange data between database formats. This file is the result of exporting data from some source database (program) to a certain exchange file format, that the target database (program) is able to import. Exchange file formats are standardized, and usually simple formats that most database programs can export to or import from. The advantage of using exchange files is that there are only a limited number of them, that the format is completely known, and that they are usually text files that can be opened and/or edited in any text editor.

Adlib is capable of converting (importing) databases from the following exchange file formats to the Adlib database format:

- Tagged ASCII (Adlib)
- DBASE III/IV (*.dbf)
- ASCII delimited (*.csv)
- ASCII fixed length
- PICA III

- MARC (general ISO 2709)
- MARC (Ocelot)
- XML
- MARC (CDS-Isis)
- Image directory
- Modes

You can also import similar formats, if just the field or record separators used in the exchange file are different.

An export job defined in Designer though, will always export to the *Tagged ASCII (Adlib)* exchange file format. (With the *Export wizard* in a running Adlib application, you may also export to *ASCII delimited (*.csv)* or *XML*.)

And then there's also the issue of the character set in which database and exchange file are encoded:

- For import, Designer automatically recognizes UTF-8 and Unicode (big endian as well as little endian) exchange files and imports the data from them, regardless of the type of the target database. For ANSI and DOS databases the following applies: if there are non-importable Unicode characters in the exchange file, they will be replaced by "?" (a question mark).
- Designer exports data from DOS, ANSI or UTF-8 encoded Unicode databases to exchange files in UTF-8.

Although Adlib does quite a lot of the converting work for you, you will have to take care of a number of things yourself. For instance, you will have to determine the target Adlib fields in which each field from the source database in the exchange file must be read in. And if different types of data are packed in a single field in the exchange file, you may want to "unpack" these subfields with an ADAPL program (which must be written first). These and other options must be set in what's called an import job. This is a settings file with the extension *.imp*. To actually start importing, you must run the import job. Similarly, you must set a few options in an export job (an *.exp* file), and run it, to actually start exporting data.

Creating, editing, saving, deleting or running import and export jobs can be done in the *Import/Export job manager* and the *Import/Export job editor*.

When you are about to import multiple exchange files in as many databases, you'll have to think about the order in which to import them, as this is not a trivial matter. A typical import of multiple exchange files (exported from Adlib databases) in as many empty

target Adlib databases with the same record number ranges, would involve the following:

1. First import the *Thesaurus* and *Persons and institutions* exchange files while only mapping the *term* field and keeping the record numbers identical. Then import the same *Thesaurus* and *Persons and institutions* exchange files again, with a full tag mapping this time, with the *term* field as the update field, without processing links and without creating new records. This is to prevent that Designer will try to create internal links during the first import run, because that would logically fail: the internal links must point to records which may still have to be imported, yet you don't want Designer to create new internally linked records with different record numbers from the ones you are importing. So the first import run creates all basic records and the second run will write the internal links in the basic records.
2. Only then import the catalogues, without processing links, whilst keeping the record numbers identical. As long as the processing of links is off, no new linked records will be created, so the order of importing, for instance, the photo and catalo databases (which are linked) doesn't matter.

Exporting and importing also offers you some related functionality, like:

- Instead of importing an exchange file, you can setup an import job so that it only reindexes all the indexes in the Adlib target database. See the Help topic on the *Import database file* option on the *Advanced* tab of the import job properties.
- You can convert DOS encoded records from an exchange file to the proper character set of the target database or an ANSI encoded exchange file to UTF-8 encoded Unicode, during import. And there are also possibilities to convert UTF-8 encoded export files to another encoding. See the Help topic *Character set conversion of your data and/or application*, for more information.

Tag files

In the old ADSETUP tool for editing applications, you could export or import so-called tag files. These were text files in which an entire application structure was described, intended to copy application definitions from one machine to another by means of a floppy disk or by e-mail through a dial-in connection, for example. If bigger storage media and faster internet connections hadn't made this functionality obsolete already, then Adlib Designer has, by making it very easy to copy application definitions or parts thereof. So Adlib Designer does not offer the possibility to export or import tag files.

See also

- Accessing the job managers and editors
- Managing import and export jobs
- Editing job properties
- Saving modifications
- Running import and export jobs

7.2 Interface functionality for import and export

§

7.2.1 Accessing the job managers and editors

To access an import or export job in Adlib Designer, you have to open the *Import job manager* or the *Export job manager*, depending on the type of jobs you are looking for. (Note that import and export jobs cannot be found with the *Application browser*.)

In the job managers the import or export jobs located in the current work folder are listed. From here, you can open an existing job (or more than one) in the *Import job editor* or *Export job editor*, create a new job, or perform some other related functionality.

Follow these steps:

1. In the main *Adlib Designer* window that opens when you start Designer, start the *Import job manager* by choosing *View > Import job manager* or clicking the button for this tool:



Or start the *Export job manager* by choosing *View > Export job manager* or clicking the button for this tool:



Note that you can only have one of the managers opened at the same time.

2. Select your work folder in a job manager window, by clicking the *Change working folder* button:



Choose the `\data` subfolder of your (copy of an) Adlib folder, since

import and export jobs are normally saved in this folder.
The present jobs in the selected folder are immediately listed.

3. The properties of an import or export job can be viewed and edited in the *Import job editor* or *Export job editor*. You can open the appropriate editor from a job manager in different ways:
 - double-click an existing job in the list;
 - right-click an existing job in the list, and choose *Open* in the pop-up menu;
 - start a new job either by choosing *File > New*, right-clicking the list and choosing *New* in the pop-up menu, or by clicking the button for it:



See also

- Managing import and export jobs
- Editing job properties
- Saving modifications
- Running import and export jobs

7.2.2 Managing import and export jobs

When you have opened the *Import job manager* or *Export job manager* (see *Accessing the job managers and editors*), you can manage the concerning import or export jobs listed.

Sort the list

You can sort the list by clicking the column header you want to sort on. Click the same column header again to reverse the sort order (ascending versus descending).

Creating a new import or export job

Depending on the job manager that you have opened, you can start a new job in the appropriate editor in the following ways:

- choose *File > New* in the menu bar;
- right-click anywhere in the list and choose *New* in the pop-up menu;
- Click the button for creating a new job:



Moving and copying

Select one or more (**Ctrl**- or **Shift**-click) jobs, right-click the selection and choose *Cut* (**Ctrl+X**) from the pop-up menu if you want to move the selected jobs, or choose *Copy* (**Ctrl+C**) if you want to copy the selected files. If you want to paste the cut or copied files in the same folder or elsewhere, then select the desired work folder, right-click the list of jobs in it (or the empty list) and choose *Paste* (**Ctrl+V**) in the pop-up menu.

Create documentation

To create documentation for a selected job, choose *File > Create documentation*, or click the button for it:



A *Documentation* window will be opened with a detailed description of the current job (all its properties and some file metadata).

The overview is nicely laid out, but if you rather have the bare XML view, you can switch to it through the *View* menu.

In either view you can print the file via the menu or the button, or save it as XML file.

You may also select the text, or part of it, with the mouse cursor, and copy (**Ctrl+C**) and paste it (**Ctrl+V**) in any text editor document.

Deleting import or export jobs

Select one or more (**Ctrl**- or **Shift**-click) jobs that you want to delete, and either choose *Edit > Delete* (**Del**), right-click the selection and choose *Delete* in the pop-up menu, or click the button for it:



Deletion of files is currently permanent: you cannot restore deleted files from Windows' recycle bin.

See also

Accessing the job managers and editors

Editing job properties

Saving modifications

Running import and export jobs

7.2.3 Editing job properties

If properties of an object are displayed in white or yellow entry fields, you can edit them. (The yellow colour is just for visual presentation, it has no special meaning.)



Just click in the entry field, and delete or type characters.

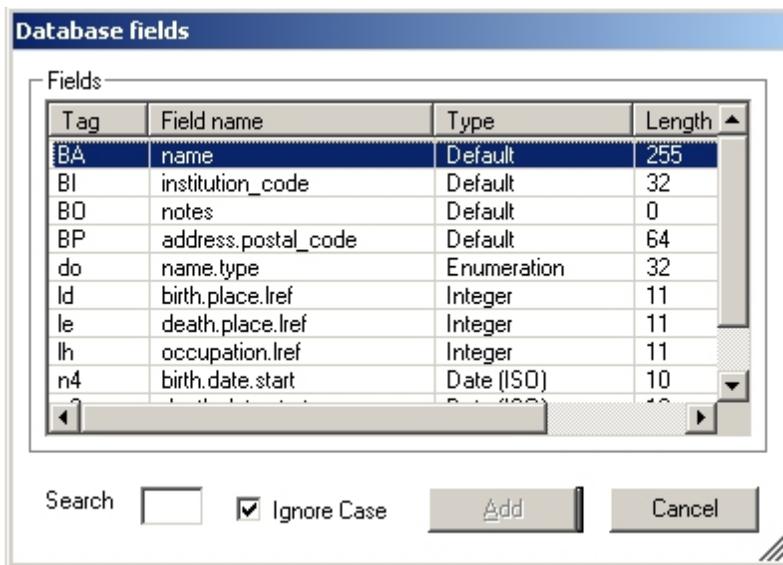
If properties of an object are displayed in greyed out entry fields, you cannot edit them: they are read-only.

Choosing a value from a list

Behind a number of entry fields, you'll see a grey button with three dots on it (see the figure above). Click this button to either open a list of files/objects that are available for the current property, to search your system for a specific file/object, or to create a new file/object at a specific location on your system. Whenever the button is available, you are advised to use it. For properties that need a tag or field or other object name (like a dataset name), the list offers all the objects that you can enter for this property, and choosing one from the list makes sure the name is spelled just the right way and that any path to the object is entered correctly.

Even though when you type values yourself, sometimes they are validated against the list that you open with the ... button, there is still a chance that by typing you enter incorrect values, either because you forget that certain values are case-sensitive or because you remembered the name of a tag incorrectly. So use the ... button whenever possible. It helps enormously to be able to just choose from the values that apply for the current property. For example, for the properties of an import job you can choose the destination *Database* name from a list of database files, and then you can choose the *Update field* from a list of (indexed) fields available in that database!

In the list window that opens when you click the ... button to search for a database field, you choose a field either by double-clicking it, or by selecting it and clicking the *OK* button (not visible in the reduced image of that window, below).



You can sort the list on any column ascending or descending by clicking the column header once or twice.

You can limit the list to all tags that begin with a certain character by typing that character in the *Search* entry field. Remove it to display the entire list again. If you mark *Ignore case*, the tags searched with the character(s) that you provided can be upper case as well as lower case.

If in the *Search* field you type a tag that does not exist yet, and you click *Add*, the new tag is forced in the field property, but no field definition is created in the concerning database, so use this option sparsely, it at all.

Change the name of an existing job

If you change the name of a job through its properties, and save the job, the new name/job will also appear in the job manager. The job will be saved under the new name; the job file with the old name is not deleted.

Moving through properties

You can easily move downwards through properties by pressing **Tab** on your keyboard, or **Shift-Tab** to move upwards.

Editing mappings

On the *Mapping* properties tab for an import job, you'll find a tag pair

mapping.

Click the *Add* button on the tab itself to add a new entry in the mapping list. Click *Delete* to delete the currently selected mapping entry.

Source	Destination	Field Name	Comments
xx	xx		

Click a cell in the mapping to edit it. In the *Source* and *Comments* cells you must type a value yourself. The cells in the *Field name* column however have a special function: clicking a cell behind a *Source/ Destination* cell pair, opens a list with all fields/tags that you can choose from. Select a field from this list, and the associated tag is automatically filled in in the *Destination* cell, next to the field name that you selected.

See also

Accessing the job managers and editors

Managing import and export jobs

Saving modifications

Running import and export jobs

7.2.4 Saving modifications

Changes* in the properties of an import or export job can be saved directly manually in its own (*.imp* or *.exp*) file from the *Import job editor* or *Export job editor* by choosing *Save* in the *File* menu or by clicking the *Save* button:



If you start running a job you'll be asked if you would like to save the job first (if the job has changed). It is mandatory to save the job before running it. Saving a job first has the added advantage that there can be no doubt later on, about which settings were used to execute the job.

You can also only save changes in selected files, by choosing *File > Save all modified objects* or by clicking the *Save all* button, in the main *Adlib Designer* window:



In the *Save objects* window that opens when you click this button, you will be presented with an overview of all the unsaved files (including import and export job files) in which you made changes, and you can determine of each of these separately whether you want to save them or not, by selecting them or deselecting them. Click *Yes* to save the selected files. *Cancel* returns you to Designer.

When you close Adlib Designer while you haven't saved all changes yet, the *Save objects* window will automatically appear, allowing you to save your work before the application is closed.

* The nature of properties forms and lists sometimes requires you to leave a property that you just edited, before you can save it. So when you have edited your last property for today, make sure you first click some other property or tab, before you save all your work.

See also

Accessing the job managers and editors

Managing import and export jobs

Editing job properties

Running import and export jobs

7.2.5 Running import and export jobs

Saving an import or export job, doesn't execute the job. You must select a job in either the *Import job manager* or *Export job manager*, or open it in the *Import job editor* or *Export job editor*, and then instruct Designer to start importing or exporting according to the settings in the selected or opened job, by choosing *File > Run* or clicking the button for it:



Every time you execute the current job, Designer will refresh any loaded adapl procedures set for this job, so that you can easily test a changed version of your newly compiled adapl by just running the job again.

Any storage adapl associated with the target database will not be executed during import via Designer (or import.exe).

If the destination file for an export job already exists, you'll be asked if it should be overwritten or not, before the export job is started.

The *Run* tab in the appropriate job editor automatically opens when you run a job. It displays some statistics about the running/ran job. The status bar in that window also indicates if the job is still running or if it has been completed. In the main Adlib Designer window a log of the operation is kept: any errors during execution of the job are also logged there.

The running of an import job by Designer, is done in the background, which allows you to continue working in Designer. You can only run one job at a time; Designer won't allow you to run multiple jobs simultaneously.

You may cancel a running job at any moment, for instance if errors occur, by clicking the *Stop import/export job* button:



The status bar indicates that the job has been aborted. Note that already exported or imported records are not made undone by stopping the job: you will be left with a partial import or export.

Note that importing from Designer is just as fast as with *import.exe*, because it uses the same code.

Export and import progress

How long exporting, or importing and indexing takes, depends on your computer system, the size of the database, the average record size, the number of indexes and links and the number of repetitions in the event of repeatable index fields.

During an import operation, the *Import job editor* constantly shows how many records have been processed (depending on how you set the *Milestone* option), the number of records processed per minute, the estimated end time, a progress bar, and some other data.

The export progress data is similar to that of importing, but there's no estimated end time or progress bar. Exporting is much faster than importing though, which makes said progress data less relevant.

Exclusive access

On the *Run* tab of an import job, you may set the *Exclusive access* option before running a job. This option specifies that Designer has exclusive access to the database. Now, locking and similar operations cannot occur, making the import process much faster.

This option is not part of a (saved) import job, it's just a temporary runtime parameter to speed up conversion by not performing any form of locking. Moreover, the option is only relevant for CBF based conversions and should be used carefully, since using it when other users are active will almost certainly result in data corruption.

Errors

Errors that occur during import are displayed on the *Run* tab of the *Import job editor*, and in the main *Adlib Designer* window (from where you can save or print them). (The old DBSETUP tool used to register these errors in a *.log* text file.) Importing will continue if errors are non-fatal. Two types of non-fatal errors can occur:

1. when a key value is longer than the index length; Designer will abbreviate the key to the destination length and index it.
2. when the program encounters a date as an index value, while the notation of the date is incorrect; Designer will not index the date. The 'incorrect' date, however, will be imported in the database anyway.

The following data about an occurring error will be reported:

- the record number;
- the tag of the field in which the error occurred;
- the occurrence of the field;
- the data itself.

See also

Accessing the job managers and editors

Managing import and export jobs

Editing job properties

Saving modifications

7.2.6 Using batch jobs

The purpose of the *Batch job manager* is to be able to sequentially run several import (or export) jobs (called a "batch") automatically. You only need to use this functionality if you have one or more batches which you'll want to run more than once.

To access the *Batch job manager*, click the *Run batch manager* button in the main *Adlib Designer* window:



Creating a new batch job

A new, empty batch job is opened automatically. You can add existing

import or export jobs to this batch in one of the following ways:

- choose *Job > Add* in the menu bar;
- click the button for adding an import or export job to the list:



Import and export jobs can normally be found in the `\data` subfolder of your (copy of an) Adlib folder.

Loading an existing batch job

If you created and saved a batch job earlier, then you can easily load it into the manager again in one of two ways.

- choose *File > Load batch* in the menu bar;
- click the *Load a batch* button:



Sort the list

The order in which import or export jobs are listed in the batch is the order in which they will be executed (from top to bottom). If you want to change the sorting of this list, simply click one of the job names to select it and then click the up or down button (repeatedly) to move the job to the desired spot. Do this for every job until the listing order is right for you.



Removing an import or export job from the batch

In the list, select the name of the job that you want to delete, and either choose *Job > Remove*, or click the button for it:



Saving a batch job

A newly created batch job or changes in the composition of an existing batch job can be saved directly in its own (`.batch.xml`) file from the *Batch job manager* by choosing *Save as* in the *File* menu or by clicking the *Save batch definition* button:



Marking dependent import or export jobs

In front of every listed import or export job in this batch, you'll see a checkbox. You should mark all jobs which are only allowed to run if the job before it has been processed successfully. All directly following marked jobs will be skipped if errors are encountered while running the preceding job in the batch.

This safety measure is especially handy when you're importing while processing links.

Running a batch job

To run a batch job, either choose *File > Run*, or click the *Run batch* button.



When running the batch job, some statistics about the running/ran job are displayed in the *Batch job manager*. In the main Adlib Designer window a log of the operation is kept: any errors during execution of the job are also logged there.

Progress of the batch job

During each running import or export job, you'll be kept up-to-date with the job processing status, with how many records have been processed (depending on how you set the *Milestone* option), the number of records processed per minute, the start time and the end time. At the bottom of the window you'll see a progress bar, and above it a box showing any occurring (error) messages.

See also

Accessing the job managers and editors

Managing import and export jobs

Editing job properties

Saving modifications

Running import and export jobs

7.3 Properties of import jobs

§

7.3.1 General

On the *General* tab, which is present when you have opened a new or existing import job in the *Import job editor*, you specify the basic properties of the current import job.

[Click here](#) for information on how to edit job properties in general.

[And click here](#) to read about how to manage import and export jobs.

On the current tab you'll find the following settings:

Job name

Enter the name this job should have (the name of the file that holds these settings). This name must comply with your operating system's requirements for file names (for DOS programs, a file name can only have a maximum length of 8 characters; Windows supports long file names). You don't have to enter an extension: Designer will fill in *.imp*.

Description

Enter a description of the import job, to clarify what it does.

Input file type

Here you indicate how the exchange file that is to be imported is organized. You can choose from:

- Tagged ASCII (Adlib)
- DBASE III/IV (*.dbf)
- ASCII delimited (*.csv)
- ASCII fixed length
- PICA III
- MARC (general ISO 2709)
- MARC (Ocelot)
- XML
- MARC (CDS-Isis)

- Image directory

You can also import similar formats, if just the field or record separators are different.

Source data file

Enter or search for the path and name of the exchange file that you wish to import. This name must comply with your operating system's requirements for file names.

Folder

This property contains the full path to the database into which you want to import data, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the path to that folder is automatically entered here. Usually, you keep your databases in one folder, the `\data` subfolder of your Adlib folder.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you wish to import the data from the exchange file into. Do not enter the extension of the file. Examples of database names are `DOCUMENT`, `COPIES`, and `Collect`.

If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) If you only want to import into one specific dataset in the database, you must select this dataset.

7.3.2 Mapping

On the *Mapping* tab, which is present when you have opened a new or existing import job in the *Import job editor*, you pair tags in the exchange file to tags in the target database, for the current import job.

Click here for information on how to edit job properties in general. On the current tab you'll find the following settings:

Tag pair list

In most cases, the field names or tags in the exchange file to be imported will differ from the field names (tags) in Adlib. With the field mapping you specify how Adlib must translate field names, by pairing source tags from the exchange file to Adlib tags.

The naming of the fields in the exchange file depends on the exchange file format (see below for a list) and on how that file was created. If you don't know the field names in this file, try to open it in a text editor to find out what the source tags are; and if you can't open it in a text editor, use the original database program. If you already chose a destination database on the *General* tab of this import job, you can easily fill in the destination tags, by clicking the ... button next to the *Destination field* column cell of the current tag pair. A list with available fields in the target database will open, for you to choose from; the destination tag or field name is automatically entered in the proper table cell.

If the names of the source tags in the exchange file correspond exactly with the destination tags in your Adlib database (e.g. because you exported from Adlib first), you can define a one-to-one translation, by providing only one special tag pair as the mapping:



To exclude certain fields from being imported if you use the mapping `** | **`, map each of those fields literally to: `<null>` (the word "null" enclosed by sharp brackets). Fields mapped to `<null>` will be ignored during import.

Examples of tag pair lists for specific exchange file formats can be found in the description of the relevant format (see the list below).

Which tags should be included in your mapping depends on which data from the exchange file you actually want to import and on the field definitions in the target database. This is especially relevant for linked fields which have associated link reference tags and possibly some merged-in fields. See the import options *Process external links* and *Process internal links*, for information about exporting and importing such fields.

If you want to import data into a field of incompatible data type, you need to process that data during import (after reading in a record from the exchange file and before writing it to the target Adlib database), via an *Adapl procedure* which must be written for this purpose.

For importing metadata from an image directory, there is a special source tag available to include in the tag pair list in the import job: `PATH`. In the exchange file, in `PATH`, the absolute path will be stored, including the file name of the imported image. (The Windows `FILE` source tag on the other hand, only stores the file name, without the path.) An example of an absolute path is: `C:\myimages\collection1\v1203dr.jpg`

In the tag pair list you can also specify field and record separators

that are different from the default separators for the current input format.

Especially for formats very similar to the ASCII delimited (.csv) exchange file format, you can specify an alternative character or string which marks the beginning and ending of field data. By default, field data in this format is enclosed by double quotes, but if the format of your exchange file uses some other character, then you can specify this here in the field mapping by providing the literal character or string as the source tag, and literally <LITERAL> as destination tag. If field data is not enclosed by any characters, then literally enter as source tag: <null>. Examples:

'	<LITERAL>
---	-----------

<null>	<LITERAL>
--------	-----------

Note that the default character to enclose field data is the double quote ("). You do not need to specify the default setting in the field mapping, as it is assumed implicitly.

Comments

You can document your mapping by adding comments to individual tag pairs. (These comments are also included in the *Documentation* that you can generate of this import job.) This is only useful if you save this import job.

Language

From 6.5.1, this *Language* option and the *Default language* option on the *Options* tab provide a way to specify the language attribute (using a standard language code*) with which an imported value (originally without language attribute) has to be stored in a multilingual target field (in an Adlib SQL or Oracle database only), during import. With the current option you specify this language per mapped field, while the *Default language* option on the *Options* tab sets the language for all multilingual fields in the target database together.

These two options always apply if your exchange file is of the Adlib tagged type, because it is mono-lingual: it cannot hold multi-lingual fields.

For Adlib XML exchange files, both options apply only to mono-lingual field values (values without language attributes): Adlib XML format can still hold mono-lingual fields, even if multilingual fields are present as well. So on import of mono-lingual fields (fields without language

attribute) into a database that does contain multilingual fields, you'll want to select the data language into which the imported values will be stored. If you do this in the field mapping, you can specify per field into which language a value must be imported. You can even repeat a target field in the mapping, provided that you specify a different language for all repetitions; this allows you to e.g. import tag `ti` into multilingual tag `ti` in the Dutch language, while importing tag `ta` (which in this example contains a translation of the title from `ti`) to the same multilingual tag `ti` in the English language.

For multilingual fields in Adlib XML type exchange files, the *Default language* and field mapping *Language* setting do not apply. This is because Adlib XML contains a language attribute per multilingual field value. When you import a record with multi-lingual fields, all multilingual values from the exchange file will automatically be copied to the target database, thereby leaving other, existing multilingual values intact. So, if a source field were to hold three values in the Dutch, English and French languages, and the target field would hold two values in the Dutch and Greek language, then the Dutch value would be overwritten, the Greek value would remain intact, and the English and French values would be added. If multilingual fields in an Adlib XML exchange file contain a flag marking the invariant language, then this flag will be imported as well.

If you use neither option, values will be added to multi-lingual fields without any language attribute; however, when a record with a multi-lingual field is written again, for instance by editing and saving it, Adlib will check if the field already has a data language attribute and if not, it will add the active data language as the attribute.

* The language code is a code put together from an abbreviation for a language and a region identifier (for more information about this, see the "Using Language Identifiers (RFC 3066)" document which you can find on the internet). The code for British English, for example, is `en-GB`.

Update option

From 6.5.0, you can indicate here that you do not want to replace the occurrences of a field, but wish to append any new occurrences from the exchange file to the existing occurrences of the field in the target record. This is useful if a record in the exchange file which you want to import contains new information in certain field occurrences, but not (all of) the old information still present in your database, while after import you want the old and the new information to be stored in separate occurrences of the field. This functionality is intended for single repeatable fields, not repeatable field groups. In the *Update option* drop-down list you have the following possibilities:

- *Overwrite*: this is the standard setting, which does nothing other than what older versions of Designer did by default: the contents of the target field occurrences must always be overwritten with the imported data.
- *Append*: all occurrences of this field will be appended as new occurrences to occurrences of this field already present in the target record. The *Delete data in existing records* option on the *Options* tab must be set to *No*, and of course *Clear database* must not be marked.
- *Append if not present*: occurrences of this field will only be appended as new occurrences to occurrences of this field already present in the target record if the contents of those new occurrences is different from the contents of already present occurrences. The *Delete data in existing records* option on the *Options* tab must be set to *No*, and of course *Clear database* must not be marked.

See also

Exchange file formats:

- Tagged ASCII (Adlib)
- DBASE III/IV (*.dbf)
- ASCII delimited (*.csv)
- ASCII fixed length
- PICA III
- MARC (general ISO 2709)
- MARC (Ocelot)
- XML
- MARC (CDS-Isis)
- Image directory

7.3.3 Options

On the *Options* tab, which is present when you have opened a new or existing import job in the *Import job editor*, you specify the basic options for the current import job.

Click here for information on how to edit job properties in general.

On the current tab you'll find the following settings:

Update field

While importing, Designer can check whether a record already occurs in the database and update it; only tags specified in the tag pair list (field mapping) of this import job, will be updated. To do this, the program has to know which field is unique for each record (e.g. a record number field). You can search for the desired (indexed) field in the destination database by clicking the ... button.

If you leave this field empty, Designer will not check whether records already exist. The records in the import file will simply be added to the database. If you leave the *Ignore priref* property unmarked too, the imported records will be given the same prirefs (record numbers) as they had in the exchange file. Any existing records with the same priref will then be completely overwritten!

If you fill in %0 (= primary reference) for the *Update field* property, Designer will check whether each specified priref in the exchange file already exists in the database. If so, the existing record will be overwritten ("updated" as it were), as far as specified in the tag pair list of this import job. If you have left the *Delete data in existing records* property (see below) set to *No*, then tags already present in the existing record will be kept intact, provided they don't occur in the exchange file: so tags (and their contents) in the database will only be replaced if they are in the exchange file too (and in the tag pair list of course).

If you enter another tag for this field than the priref, there must be a text (term) index for the field (this index need not be unique). Designer will check whether the specified tag value (the contents of the tag) in the exchange file already exists in the database. If so, the corresponding record(s) will be overwritten. If you have set the *Delete data in existing records* property to *No*, tags in the existing record(s) will remain, provided they don't occur in the import file record.

You can further specify the updating of target records, pertaining to occurrences of tags, by setting the *Update option* on the *Mapping* tab to *Append* or *Append if not present*.

Please note that Designer will only add records (as in: enlarging the total number of records in the dataset) if you have marked the *Add new records* property (see below) for the above situations (where *Update tag* is filled in). If you unmark this property, records, or parts of

them, can only be replaced.

If the update field is a multi-lingual field (in an Adlib SQL or Oracle database), all translations of the present data will be checked for the update value; it is not possible to have Adlib check only one of the translations. Adlib first checks all records in a language 1, then all records in a language 2, etc. If the update value is found, the record will be updated.

Update language

From 6.5.1, you can use a multi-lingual field as the update tag in an import job, and specify a language in which to search. If you do not specify a language for a multi-lingual field, all language values of the update tag will be searched.

The *Update language* is optional, but when used it should be filled with a standard language code*. You can use this option with all exchange file formats.

* The language code is a code put together from an abbreviation for a language and a region identifier (for more information about this, see the "Using Language Identifiers (RFC 3066)" document which you can find on the internet).

Add new records

If you have entered a value for *Update field*, you can also decide whether it will be possible to add new records too, instead of only replacing records. Mark this option if you do want adding to be possible.

Delete data in existing records

If you have filled in a value for *Update field*, you can also specify which tags must be deleted when a record is updated. You can choose three settings:

- *No* - All tags from the exchange file record (if specified in the mapping) replace any existing tags; all other existing tags remain intact.
- *Yes, delete all fields* - All tags from the exchange file record (if specified in the mapping) replace any existing tags; all other existing tags are deleted.
- *Yes, only mapped fields* - All mapped fields are deleted, and all tags from the exchange file record (if specified in the mapping) will be added to the record; all other existing tags will remain intact.

Process external links

You can use this property to specify whether Designer is to automatically update external links to linked records. An external link is a link from a record in the current database to a record in another database (like the link to a name record in the PEOPLE database, from within the *Author* field of a book title record in the DOCUMENT database), while an internal link is a link from a record in the current database to another record in the current database (like you have in the Thesaurus, for instance), as defined under the *Internal links* node in the database tree in the *Application browser*.

Mark this checkbox, and for a linked field it will be checked whether the imported field value occurs in the linked-to field anywhere in the linked database. If not, a linked record for that value does not exist yet, and a new record for that value will be created in the linked database (and the record number of the new linked record will be retrieved - this is called resolving - and imported into the linkref tag* of the linked field in the target database). That means that the *Allow creation of new linked records* property of these linked fields in the database setup, must be set to true, or that you must set *Forcing always allowed* in this import job (see below). If the field value does already occur in the linked-to field in the linked database, the record number of the relevant linked record will be resolved and imported into the linkref tag of the linked field in the target database.

* Most, but not all, externally linked fields have associated link reference tags. If no linkref tag is present, processing a link just won't retrieve a record number of the linked record; this will not cause problems.

If you unmark the *Process external links* option, linked fields and their values, and/or link reference tags and the record numbers they contain, will be imported, but during this import there will be no check whether the linked records for any imported field values actually exist. This means that if this destination field has a link reference tag, this tag will remain empty (if it was empty in the exchange file) while the linked field is filled, or the linkref tag will be filled with the record number from the exchange file (which may or may not be correct: without resolving you cannot always be sure of this). Note that any faulty link reference tags cause problems in your data and indexes, while missing link reference tags may cause errors if you try to export them again.

Any non-preferred terms in the exchange file will be substituted by the defined preferred terms during import for all externally linked fields in the primary destination database, if *Process links* is marked.

Note that from Designer 6.5.1, the previous *Process links* option has been split up in the two options *Process external links* and *Process*

internal links, to be able to separate the processing of links. This can result in faster importing if the processing of one of both types of links is not required. Normally though, you would mark both options.

In the overview below, all export/import combinations pertaining to links processing are explicated. To keep the text as straight forward as possible, we assume here that simply all tags are included in the field mapping of the import job; but consider that for linked fields it is good not to include merge field tags if you process links during import, because including those tags would store their values in the target fields. Yet, actual merged-in field values in the target database will automatically be retrieved from the linked record later whenever a record is opened, while those values won't be stored in the destination fields. So, excluding merged-in field tags from the import job (with links processing on) keeps their counterparts in the target database empty (as they should be), and makes your import faster.

Export - links processing option	Import with external links processing on/off
<p><i>Export with links</i>: only linked field values</p> <p>Of linked fields, this will export any linked data to the exchange file, meaning the tag of the linked field (in the primary database) and its resolved contents: the field value copied from the linked record, and also merged tags defined for this linked field, and their contents.</p>	<p><i>On</i>: of linked fields, the field values will be imported and resolved, so that the linkref tags in the target database will be filled with actual record numbers of the linked records.</p> <p>Merged-in field values that were exported with the source record are imported as well, the actual merged field values for the linked field are not retrieved during import: only after import, when you edit and save a record, will actual merged-in field data be retrieved for linked fields.</p> <p>Automatic non-preferred term substitution can take place.</p> <hr/> <p><i>Off</i>: of linked fields, only the field values will be imported. Adlib will not resolve the linked field values, meaning: it won't look up those values in the linked database to retrieve the record numbers of those linked values, so you won't be sure if linked records actually exist for the imported values, and the linkref tag of the linked field will remain empty; and because of the empty linkref tag, the linked field value will not be indexed, so you won't be able to search</p>

	<p>on this value afterwards via its index/access point.</p> <p>After import, non-resolved field values will only be resolved and indexed by opening all relevant primary records for editing in Adlib and saving them.</p> <p>Non-preferred term substitution will not take place.</p>
<p><i>Export full record</i>: field values plus linkrefs</p> <p>This option does the same as <i>Export with links</i>, but adds the forward reference tag and its content: the record number of the linked record that holds the field value for this field.</p>	<p><i>On</i>: of linked fields, the field values will be imported and resolved, so that the linkref tags in the target database will be filled with actual record numbers of the linked records, and these record numbers are not necessarily the same as the record numbers in the linkref tags in the exchange file. The linkref values from the exchange file won't be imported into the target database at all. Merged-in field values that were exported with the source record are imported as well, the actual merged field values for the linked field are not retrieved during import: only after import, when you edit and save a record, will actual merged-in field data be retrieved for linked fields. Automatic non-preferred term substitution can take place.</p> <p><i>Off</i>: of linked fields, the field values and the associated record numbers of those linked records (in the linkref tags) will be imported. Adlib will not resolve the linked field values, meaning: it won't check whether the imported linked record numbers are correct, so it is possible that corrupt links will be created.</p> <p>Non-preferred term substitution will not take place.</p>
<p><i>Export without links</i>: only linkrefs</p> <p>Of linked fields, this will only</p>	<p><i>On</i>: no linked field values and no linkrefs will be imported. A linked field value is required for resolving the link (retrieving the relevant record number); since a linked field value is not present in the</p>

<p>export a reference to linked data, meaning the forward reference tag and its content: the record number of the linked record that holds the field value for this field.</p>	<p>exchange file, there is no way to validate the record number from the linkref tag in the exchange file, so the linkref tag in the target database will remain empty. Non-preferred term substitution will not take place.</p>
	<p><i>Off:</i> of linked fields, only linkrefs will be imported. Since Adlib won't try to resolve the link, the record number in the linkref tag will just be imported into the linkref tag in the target database. Non-preferred term substitution will not take place.</p>

The general rule of thumb is that if you exported data with links processing on, then you should also import that data with links processing on. And the reverse is also true: if you exported data with links processing off, then you should also import that data with links processing off.

Process internal links

You can use this property to specify whether Designer is to automatically update internal links to linked records. An internal link is a link from a record in the current database to another record in the current database (like you have in the Thesaurus, for instance), as defined under the *Internal links* node in the database tree in the *Application browser*, while an external link is a link from a record in the current database to a record in another database (like the link to a name record in the PEOPLE database, from within the *Author* field of a book title record in the DOCUMENT database).

Mark this checkbox, and for a linked field it will be checked whether the imported field value occurs in the linked-to field anywhere in the current database. If not, a linked record for that value does not exist yet, and a new record for that value will be created in the current database (and the record number of the new linked record will be retrieved - this is called resolving - and imported into the linkref tag of the linked field in the current database). That means that the *Allow creation of new linked records* property of these linked fields in the database setup, must be set to true, or that you must set *Forcing always allowed* in this import job (see below). If the field value does already occur in the linked-to field in the current database, the record number of the relevant linked record will be resolved and imported into the linkref tag of the linked field in the current database.

Since internal links may define preferred term relations, of course no non-preferred term substitution will take place during import.

If you unmark the *Process internal links* option, linked fields and their values, and/or link reference tags and the record numbers they contain, will be imported, but during this import there will be no check whether the linked records for any imported field values actually exist. This means that if this destination field has a link reference tag, this tag will remain empty (if it was empty in the exchange file) while the linked field is filled, or the linkref tag will be filled with the record number from the exchange file (which may or may not be correct: without resolving you cannot always be sure of this). Note that any faulty link reference tags cause problems in your data and indexes, while missing link reference tags may cause errors if you try to export them again.

Note that from Designer 6.5.1, the previous *Process links* option has been split up in the two options *Process external links* and *Process internal links*, to be able to separate the processing of links. This can result in faster importing if the processing of one of both types of links is not required. Normally though, you would mark both options.

In the overview below, all export/import combinations pertaining to links processing are explicated. To keep the text as straight forward as possible, we assume here that simply all tags are included in the field mapping of the import job; but consider that for linked fields it is good not to include merge field tags if you process links during import, because including those tags would store their values in the target fields. Yet, actual merged-in field values in the target database will automatically be retrieved from the linked record later whenever a record is opened, while those values won't be stored in the destination fields. So, excluding merged-in field tags from the import job (with links processing on) keeps their counterparts in the target database empty (as they should be), and makes your import faster.

Export - links processing option	Import with internal links processing on/off
<p><i>Export with links</i>: only linked field values</p> <p>Of linked fields, this will export any linked data to the exchange file, meaning the tag of the linked field (in the current database) and its resolved contents: the field value copied from the linked record, and also</p>	<p><i>On</i>: of linked fields, the field values will be imported and resolved, so that the linkref tags (if present) in the target database will be filled with actual record numbers of the linked records. Merged-in field values that were exported with the source record are imported as well, the actual merged field values for the linked field are not retrieved during import: only after import, when you edit and save a</p>

<p>merged tags defined for this linked field, and their contents.</p>	<p>record, will actual merged-in field data be retrieved for linked fields. Non-preferred term substitution will not take place.</p> <p><i>Off:</i> of linked fields, only the field values will be imported. Adlib will not resolve the linked field values, meaning: it won't look up those values in the current database to retrieve the record numbers of those linked values, so you won't be sure if linked records actually exist for the imported values. If the linked field has a linkref tag, it will remain empty, and because of that the linked field value will not be indexed, so you won't be able to search on this value afterwards via its index/access point. If, on the other hand, the linked field has no linkref tag, as is often the case in applications older than version 4.2, the linked field value will be indexed normally.</p> <p>After import, non-resolved field values will only be resolved and indexed by opening all relevant records for editing in Adlib and saving them. Non-preferred term substitution will not take place.</p>
<p><i>Export full record:</i> field values plus linkrefs</p> <p>This option does the same as <i>Export with links</i>, but adds the forward reference tag and its content (if present): the record number of the linked record that holds the field value for this field.</p> <p>Note that internally linked fields in applications older than version 4.2 often have no link reference tags.</p>	<p><i>On:</i> of linked fields, the field values will be imported and resolved, so that the linkref tags (if present) in the target database will be filled with actual record numbers of the linked records, and these record numbers are not necessarily the same as the record numbers in the linkref tags in the exchange file. The linkref values from the exchange file won't be imported into the target database at all.</p> <p>Merged-in field values that were exported with the source record are imported as well, the actual merged field values for the linked field are not retrieved during import: only after import, when you edit and save a</p>

	<p>record, will actual merged-in field data be retrieved for linked fields. Non-preferred term substitution will not take place.</p>
	<p><i>Off:</i> of linked fields, the field values and any associated record numbers of those linked records (in the linkref tags, if present) will be imported. Adlib will not resolve the linked field values, meaning: it won't check whether any imported linked record numbers are correct, so it is possible that corrupt links and index entries will be created. Non-preferred term substitution will not take place.</p>
<p><i>Export without links:</i> only linkrefs</p> <p>Of linked fields, this will only export a reference to linked data, meaning the forward reference tag and its content: the record number of the linked record that holds the field value for this field. Note that internally linked fields in applications older than version 4.2 often have no link reference tags.</p>	<p><i>On:</i> no linked field values and no linkrefs will be imported. A linked field value is required for resolving the link (retrieving the relevant record number); since a linked field value is not present in the exchange file, there is no way to validate the record number from the linkref tag in the exchange file, so the linkref tag in the target database will remain empty. Non-preferred term substitution will not take place.</p> <p><i>Off:</i> of linked fields, only linkrefs (if present) will be imported. Since Adlib won't try to resolve the link, the record number in the linkref tag will just be imported into the linkref tag in the target database. Non-preferred term substitution will not take place.</p>

The general rule of thumb is that if you exported data with links processing on, then you should also import that data with links processing on. And the reverse is also true: if you exported data with links processing off, then you should also import that data with links processing off.

Clear database

You can use this option to determine whether Designer must clear the target database before importing the exchange file. So, if this option is marked, then the entire database will be emptied!

Forcing always allowed

If you mark this option, records will be forced into a linked database, regardless of how the *Allow creation of new linked records* property of any linked fields in the target database has been set. For this, the *Process links import job* option (see above) must, of course, be selected.

Ignore priref

If a database is divided up into datasets, then the record numbers (prirefs) determine in which dataset records belong (since every dataset consists of database records in a specified range of record numbers).

If you instruct designer to ignore prirefs from records it imports (mark this option), then the program will assign new prirefs itself, starting from the lower limit record number of the dataset into which the data is imported if this dataset is empty. However, if there already exist records in the dataset into which you are importing, then new record numbers will follow the highest existing record number. And if during import the dataset record number upper limit is reached, Adlib will search for any free record numbers from the lower limit of the dataset.

If you leave this option unmarked, the imported records will be saved with their record numbers from the exchange file, and overwrite any existing records with those numbers; importing might then also occur outside the target dataset range, depending on the record numbers in the exchange file.

Milestone

Enter a value to instruct Designer how often it must provide you with information about the progress of the import operation. For example, with the value 100, this happens every 100 records (which setting is advisable for large import jobs). A larger value will speed up the import process somewhat.

ADAPL procedure

Here you specify which ADAPL procedure (if any) Designer must carry out after reading in a record from the exchange file and before writing it to the target Adlib database. This import adapl is not carried out for linked records that are created during importing.

A *Storage procedure* that may be linked to the database is never

carried out during the import with Designer.

For the programmatic design of an import adapl, you should take the following into consideration:

- you can use the `ERRORM` statement, but whenever this statement is executed during import, the message will be written to an error log file with a name formatted like `<importjob_name>.imp.err`, and won't be displayed on screen.
- The adapl will be executed per record, after the field mapping has been applied. This means that you will have to use the target field tags in your adapl to request the contents of fields from the exchange file.
- You can write to the target tags as well. The field definitions in the target database do apply, so you can't write to a second occurrence of a field tag if that field is not repeatable. This applies to all tags in the target database, not just the ones mapped from the exchange file.

You can write a value to a specific language of a multilingual field, via the syntax: `<tag>[<occ>, 'language code'] = <tag or value>`, for example: `te[1, 'en-US'] = te[1, 'en-GB']` to address the English-US attribute of the first occurrence of a multilingual Term field and copy to it the value from the already mapped first English-Great-Britain Term field occurrence. The occurrence number is mandatory. If you also want to make the target language the invariant language, you can do so by inserting a hash character in front of the target language code, for example: `BA[1, '#fr-FR'] = BA[1, 'fr-FR']` to set the French language value as the invariant one.

If mono-lingual (without language attribute) data from the exchange file all needs to be assigned the same language attribute in the target database, a more efficient solution is to use the *Default language* option (and possibly the *Make this the invariant language* option as well) below: no need to program that in an adapl.

- You can also write to tags which have not been defined in the target database. In this case you can always write to multiple occurrences.
- If you'd like to know if a field does not appear in the exchange file or is empty in the exchange file - you cannot separate these situations - then use `if (tA = '') {...}` in which you replace `tA` by the actual tag.
- You can use FACS to read and write in other databases than the target database.

- Use the `VAL` function to convert numbers appearing in tags (even temporary ones) from text strings to numerical values if you want to perform some calculation with those numbers.
- Data dictionary field group functionality like adding or sorting group occurrences, with the exception of `REPCNT`, is switched off during import of data.

In Adlib Designer versions older than 7.1, adapls which have been compiled in debug mode can only be used as *ADAPL procedure* here if you execute this import job with *import.exe*: said versions of Adlib Designer did not support adapl debugging during import. From Designer 7.1 though, adapls which have been compiled in debug mode are supported as *ADAPL procedure* here by both *import.exe* and the Adlib Designer import functionality.

Record owner

One of the possible authorisation mechanisms for a database in Adlib is the *Authorisation type* called *Rights*. With the *Rights* method, the initial creator of a record, the so-called *Record owner*, always has full access to that record, and it is this creator and he or she alone that is allowed in this record to set and change which users have which access rights, or transfer the record to another record owner. The user name of the creator of a record will be stored automatically; this name is determined through the login of the current user.

If this type of authorisation has been implemented in the target database into which you are about to import your exchange file, you may need a way to set the record owner name for newly created records without a current record owner, since Adlib would otherwise make the person who does the importing the record owner of the new records. You can do this with the *current* option: just enter the (login) name of intended record owner of newly created records in the target database, which do not have a record owner in the exchange file. Records in the exchange file which already have a record owner will not be assigned the new record owner, nor will in the target database existing records with a record owner be assigned the new record owner, if those existing records are only updated during the import - so the record owner tag won't be updated.

If newly created records without a current record owner should remain without record owner, then literally enter `<null>` for the *current* option.

Leave this option empty if the target database has no *Record owner* field.

The setup of this type of authorisation can be found on the *Database properties* tab of a database, where a *Record owner field* is

implemented for the *Authorisation type: Rights*. Authorisation fields must also be present on detail screens.

Default language

From 6.5.1, this *Default language* option and the *Language* option on the *Mapping* tab provide a way to specify the language attribute (using a standard language code*) with which an imported value (originally without language attribute) has to be stored in a multilingual target field (in an Adlib SQL or Oracle database only), during import. With the current option you specify this language for all multilingual fields in the target database together, while the *Language* option on the *Mapping* tab allows you to specify that language per mapped field.

Use the current option if your exchange file contains values in only one (non-explicit) language; then you do not have to specify a language for every multilingual target field in your field mapping separately. Just set the desired target language of all multilingual fields at once in the *Default language* option. (By the way, no invariancy flag will be added to this data language, unless you check the *Make this the invariant language* option below.)

If you use neither option, values will be added to multilingual fields without any language attribute; however, when a record with a multilingual field is written again, by editing and saving it, Adlib will check if the field already has a data language attribute and if not, it will add the active data language as the attribute.

For multilingual fields in Adlib XML type exchange files, the *Default language* and field mapping *Language* setting do not apply. This is because Adlib XML contains a language attribute per multilingual field value. When you import a record with multilingual fields, all multilingual values from the exchange file will automatically be copied to the target database, thereby leaving other, existing multilingual values intact. So, if a source field were to hold three values in the Dutch, English and French languages, and the target field would hold two values in the Dutch and Greek language, then the Dutch value would be overwritten, the Greek value would remain intact, and the English and French values would be added. Any existing invariancy flags will be imported too.

Both language options, plus the *Make this the invariant language* option, do apply for all mono-lingual fields (without a language attribute) in Adlib XML type exchange files; and these options always apply if your exchange file is of the Adlib tagged type, because that is mono-lingual by definition.

Any default language you set here, will also be used by an import adapl whenever the ADAPL code assigns a value to the tag of a multilingual field without specifying the language to write to. If you do

specify the target language in the value assignment, it will override the default language.

* The language code is a code put together from an abbreviation for a language and a region identifier (for more information about this, see the "Using Language Identifiers (RFC 3066)" document which you can find on the internet). The code for British English, for example, is en-GB.

Make this the invariant language

Make the *Default language* the invariant language by marking this checkbox. Prior to Designer 7.1, you could only add the invariancy flag manually when editing a record, per multilingual field, via the *Edit multilingual texts* window (to be opened by right-clicking the field) or you could import multilingual Adlib XML exchange files including any invariancy flags already present in the data.

The (optional) invariancy flag for a value in a particular language allows the user to see this value in all other data languages of the field as well (as long as it's empty), to ease translation or to always have data present in the field, even if it concerns data in the wrong language.

Import jobs with the *Make this the invariant language* option can also be run by the 7.1 version (or higher) of *import.exe*.

7.3.4 Advanced

On the *Advanced* tab, which is present when you have opened a new or existing import job in the *Import job editor*, you may specify some special properties of the current import job, that are usually best left to their default settings though.

Click here for information on how to edit job properties in general. On the current tab you'll find the following settings:

Input buffer size

The input buffer is memory space allocated by Adlib to read in data from the exchange file. Each time the exchange file is accessed for this purpose this amount of data is being read in. The bigger you set this input buffer, the less disk I/O (reading and writing actions) need to take place, and the faster importing will be. Therefore, by default this size is set to *32000*, for maximum performance. Only if your computer is very old and has very little memory, you may want to choose a lower setting.

Every time data from the exchange file is read in, minimally one field is read in. If the input buffer size is smaller than a particular field, the input buffer will be resized dynamically and automatically to read in

the entire field. On the other hand, a full input buffer may contain multiple records too.

Default word-wrap value

With this option, you can indicate whether you want to import the contents of long fields into one occurrence of the destination field (set the default word-wrap value to zero), or whether you want to spread such contents over multiple occurrences of the destination field (set the default word-wrap value to the maximum length that any destination field occurrence may have).

In most cases you will want to leave this setting to zero, so that the contents of long word-wrapped fields is imported into similar word-wrapped destination fields. More specifically, this setting at zero sets the source field length to the destination field length as defined in the data dictionary, during importing.

If the destination field length in the data dictionary is shorter than the character string you import, information at the end of the string will be lost.

Import database file

With this option you set whether with this import job you want to import records from the exchange file into the destination database. Since most import jobs are about importing data, you'll probably want to leave this option selected.

At first it may seem odd to have an option for this: aren't all imports about importing data? Well, most are: the only exception is when you only want to reindex your indexes (see the option below) with the help of a (dummy) exchange file, but do not want to add or change anything to your database. This is an alternative use for import jobs as reindex jobs, in which nothing is imported; then deselect this option, and unmark the *Reindex automatically* option too.

Reindex automatically

For small to normal import jobs you can let the import job simultaneously (per imported record) reindex all appropriate indexes, while the database is being filled: then unmark the *Reindex automatically* option, and of course the *Import database file* option must be marked.

For a lot of exchange files it is very important to unmark this option! For instance, for importing files that contain records with internal links you must definitely deselect the *Reindex automatically* option, because processing those links is impossible if not all indexes are updated continuously.

You can also use this unmarked option to reindex all indexes without

filling the database; of course then you need to unmark the *Import database file* option. And then the *Clear database* option on the *Options* tab, needs to be unmarked too. (You do need a dummy exchange file for this, even though you don't fill the database.)

Mark this *Reindex automatically* option though to reindex only the *preref* index and other unique indexes during import, and reindex all other indexes only after the database has been filled, to speed up very large import jobs. But make sure this database has no internal links, otherwise unmark this option.

Note: the setting of this option appears illogical, but you really have to deselect this option to activate automatic reindexing of all indexes during import.

Encoding

This option specifies the encoding in which this import job (the *.imp* file) must be saved. This is only indicative of the encoding of the import job, not of the exchange file nor of the target database. You can choose between *Oem*, *ANSI* and *UTF8*. Usually, *ANSI* is all you need. Only if text, file names or source field names in this import job contain exotic characters, you need Unicode in *UTF-8* encoding. However, a bug in Designer versions older than 7.1.13318.8 caused the *UTF8* option to store the import job in *ANSI* encoding anyway, thereby corrupting any exotic characters in the import job. So if you need *UTF-8* encoding for your import job, you require Designer 7.1.13318.8 or higher: you'll find that the encoding no longer has to be specified because it now always defaults to *UTF-8* (which suits all needs).

7.4 Properties of export jobs

§

7.4.1 General

On the *General* tab, which is present when you have opened a new or existing export job in the *Export job editor*, you specify all properties of the current export job.

[Click here](#) for information on how to edit job properties in general. And [click here](#) to read about how to manage import and export jobs. On the current tab you'll find the following settings:

Job name

Enter the name this job should have (the name of the file that holds these settings). This name must comply with your operating system's requirements for file names (for DOS programs, a file name can only have a maximum length of 8 characters; Windows supports long file names). You don't have to enter an extension: Designer will fill in *.exp*.

Description

Enter a description of the export job, to clarify what it does.

Folder

This property contains the full path to the database you want to export, without the name of the file. You don't have to fill in this property manually: just select a database in the next option, and the path to that folder is automatically entered here. Usually, you keep your databases in one folder, the `\data` subfolder of your Adlib folder.

Database & Dataset

First, enter or search for the name of the existing database (an *.inf* file) that you wish to export. Do not enter the extension of the file. Examples of database names are `DOCUMENT`, `COPIES`, and `Collect`. If the database that you select has datasets defined for it, these datasets will be listed in the *Dataset* drop-down list. (Some databases, like the thesaurus, have no datasets.) If you only want to export a single dataset from the database, you must select this dataset.

(Destination) File

Enter or search for the path and name of the exchange file that this export job will create when you run it (this is the file that will contain the exported data). This must comply with your operating system's requirements for file names. The exchange format of every Designer export job is Tagged ASCII (Adlib); you cannot choose another format. The commonly used extension of such a file is *.dat*, but you can also enter another extension. (Designer does not automatically add an extension to your file name when it creates the exchange file, so you have to provide one yourself.)

Milestone

Choose how often Designer should provide you with information about the progress of the export operation when you run this export job. With the value `10`, this happens every 10 records. A larger value will speed up the export process somewhat.

From date

Choose the record entry/modification date from which this export job may export the data from the selected dataset to the exchange file. This is the date on which a record was last modified. So all data from the selected dataset, entered or modified *after* this date, will be exported.

Link processing

You can use this option to determine whether Designer should include links to linked records in the exchange file. From Designer 6.5, you can choose from: *Export without links*, *Export with links*, and *Export full record*. In 6.1 there's just a *Process links* checkbox.

If you select *Export with links* (or mark the *Process links* checkbox in Designer 6.1), on running the export job, Designer copies any linked data to the exchange file, meaning the tag of the linked field (in the primary database) and its resolved contents: the field value copied from the linked record, and also merged tags defined for this linked field, and their contents. (If you process links during export it doesn't matter whether linked fields have forward reference tags or not.)

Choosing *Export full record* does the same as *Export with links*, but adds the forward reference tag and its content: the record number of the linked record that holds the field value for this field. This produces an exchange file which contains linked data as well as the forward references (besides all primary data of course). This option is used for certain advanced conversion tasks, for which some linked fields need some post-import processing in which additional information about those linked fields is required.

If you select *Export without links*, in the exchange file Designer inserts only a reference to linked data, meaning the forward reference tag and its content: the record number of the linked record that holds the field value for this field. For this to work, all linked fields in the records that you export, must have a forward reference tag. If this is not the case (when a linked field is linked on a term value), then the linked field without the forward reference tag will be treated as if you selected *Export with links* for it, thus exporting the tag of the linked field and the field value copied from the linked record. (NB In existing Adlib applications there may be some linked fields that do not have forward reference tags.)

Internal links (like broader and narrower terms in the thesaurus) are treated the same as external links. Note that from Adlib 5.0.1 internal links may have forward reference tags too.

In general it is recommended that you select *Export with links*, because if you don't and you re-import the exchange file later on,

making the wrong settings in the import job may result in chaos in your databases. Processing the links is just more save.

7.5 Exchange formats

§

7.5.1 Tagged ASCII (Adlib)

An Adlib tagged file (*.dat*) contains the data from every record of a certain database or dataset in readable form, but contains no index information. Its main purpose is allowing the exchange of Adlib databases between different Adlib versions or applications.

File layout

Each line begins with a tag (field label) of two or more characters, then a space, followed by the field contents. The tag begins with a percentage sign or a letter, and further consists of letters, digits or an underscore. Adlib makes a distinction between upper case and lower case letters. The field occurrence is closed with a new-line code. The order of fields is not important. Per record, a field may be repeated a maximum of 32767 times, and a record may contain a maximum of 32767 fields. Therefore, both the field contents and the record contents are variable. Repetitions of a field must be on consecutive lines. With repeated fields, the tag name is not necessary on the second and subsequent lines. There is no maximum record length.

The tag %0 is reserved for the record number (the primary reference: "priref"). The priref must be an integer and may not occur more than once in each record. If several records have the same record number, only the last one will appear in the database. If the priref in the ASCII data record is 0, Designer will allocate the next free record number to that record.

Between the records (the record separator), and at the end of the file, there is a line that only contains two asterisks. The number of records may not exceed 2,147,483,647.

Example:

```
%0 134
%1 Brown's Print Shop
%2 Brown, J.
```

%3 DE236JX
%4 Derby
A1 131 Bonsall Avenue
A3 England
A4 01510 - 752582
A5 PRINTERS
A9 Ron

Conversion

An Adlib tagged file can be imported, and Adlib can export to such a file. Most other database programs cannot read this type of file directly.

If the tag names in the exchange file are the same as the tag names in the database to which you want to import, you don't have to enter the full tag mapping for the import job. You just enter ** and ** once, as source/destination tag pair. Adlib will then use the tag names from the exchange file, when importing.

For exporting to an Adlib tagged file, the tags from the database will automatically be copied.

7.5.2 DBASE III/IV (*.dbf)

DBASE is a popular database program for PCs. The way in which this program stores databases has been adopted by a large number of suppliers.

File layout

As a rule, a DBASE file has the extension *.dbf*. These are files that cannot be read as text. All fields have a name that may consist of a number of characters. The field length is fixed, with the exception of the so called "memo" fields, of which there may be one in each record. Adlib does not read memo fields in. Repeated fields do not occur in these files.

Conversion

With Designer, DBASE files can only be imported, not exported; export is not possible because this file format is not suitable for repeated fields. Memo fields are not imported.

Field names in a DBASE file are likely to differ considerably from field names in Adlib. For the tag mapping in an import job, you enter the

names of the DBASE fields as left-hand part (source) of the tag pair; Adlib tags are the destination tags.

Example of a tag pair definition:

ADDRESS	ad
CITY	pl
NAME	na
POSTC	pc

7.5.3 ASCII delimited (*.csv)

ASCII delimited (a.k.a. comma delimited) is a much used exchange file format that is used and recognized by many database programs.

File layout

An ASCII delimited file is a text file with the extension `.csv`. All fields of a record are on one line, separated typically by commas*. If commas (or the separator character that you specify) or `cr/lf` (carriage return and line feed) characters occur in a field, that field must be embedded in double quotes. Double quotes that occur in a field must be escaped by being doubled: `""`, but when you export from Adlib this escaping is automatically applied, so you can use double and single quotes in field contents in your application normally. The fields do not have a name; instead they have a serial number, the first field in each line being 01, the next 02, and so on. The field sequence is always the same. Repeated fields do not occur.

Example:

"954",214,"libraries, museums and conservation"

"9541",12,"cultural organization"

"9542",431,"libraries, public libraries"

"9543",521,"archives (public)"

"9544",3254,"museums"

"9545",2345,"art galleries, art libraries, etc."

"9546",8532,"botanical gardens and zoos"

ASCII delimited files may also have a first line that does not contain field values, but the names of the exported fields, e.g.

"author","copy_number","title". This is for instance the case when you export to `.csv` file using the *Export wizard* in an Adlib application. But these field names are only included to document the meaning of the exported fields, they have no relevance for the tag mapping, as

explained below.

Do note that such a first line is imported as a record too! So before importing a .csv file, you should open it in a text editor and remove any first line containing field names.

Conversion

With Designer, these files can only be imported, not exported; export is not possible because this file format is not suitable for repeated fields. But you *can* export to this format from an Adlib application; only the first occurrence of a repeated field will be exported.

This type of exchange file has no field names. Instead, the fields are numbered. For the tag mapping in an import job, you enter the serial numbers of the fields in the import file as source tags, and the Adlib tags as destination tags.

Example of a tag pair definition:

01	dr
02	di
03	na
04	ve
05	si
06	ty
07	da
08	de

* You may specify custom field and record separator characters, if the exchange file format requires it.

7.5.4 ASCII fixed length

Fixed length ASCII is a much used type of exchange file that is used and recognized by many database programs.

File layout

There is no naming convention for this type of file, but it is a text file. All fields in a record are placed on one line, and each field has the maximum length reserved for that field. If that space is greater than the information in the field, the remaining space will be filled with blank spaces. The fields are indicated by the column position within

the file. The field sequence is always the same. Repeated fields do not occur. Before importing, Adlib has to know the length of each field.

Example of a tag pair definition:

1 10	T1
11 5	T2
35 15	T6

The first number in the source "tag" is the column position in the fixed length record, the second indicates the length of the field. Fields may therefore be skipped, by not providing a tag pair for a source column position.

Conversion

With Designer, these files can only be imported, not exported; export is not possible because this file format is not suitable for repeated fields.

7.5.5 PICA III

Many libraries, particularly in the Netherlands and Germany, use PICA (Project-Integrated Catalogue Automation) GGC (Shared Automated Catalogue System) for cataloguing their collection.

File layout

PICA is built up out of so called KMC codes. A PICA tag, or KMC code, consists of four digits that indicate the beginning of a certain field. Each separate field has its own KMC code. Example:

SET: S2 [167] TTL: 161 PN: 080752306 PAG: 01N

0200:1730:11-04-91 0210:1006:28-10-93 12:02:23 0230:9999:99-99-99

0500 Abx

1100 1989 \$ 1989-...

1500 /1eng

1700 /1uk

1800 f

3121 !095898263!@ Dutch Centre for Public Libraries and Literature, The Hague. Audiovisual Media Department.

3141 <10>@Info-AVM

4000 @Information on audiovisual media / [\$3121]
 4025 Issue 164 (1989) - 167 (1989) ; vol.15 (1990) + ...
 4030 The Hague : NBLC
 4062 30 cm
 4208 Appears 10x a year, issues 5 and 10 are cumulative
 7001 25-09-92 : y1vg
 4220 /b1991-
 7100 0709#019 Inf @ f
 7800 148393950

Where:

0500	a KMC code of which the 1st letter determines the type of document (e.g. A=avm, B=book, E=microform, M=printed music, S=software), the 2nd letter determines whether the document is monographic or serial, and the 3rd position signifies the status of the catalogue entry
1100	KMC for the year of publication
1500	language in which the document is published (/1 = language in which it is written, /3 = language from which it was translated)
1700	country of publication (/1 = country of publication, /2 = country of 2nd imprint)
1800	frequency of publication (a = daily, c = weekly, f = monthly)
30##	authors
31##	corporate authors
4000	title space followed by an '/' and author's statement
402#	edition / issue
403#	imprint (place of publication : publisher)
4062	format
42##	annotation
70##	copy details
71##	shelf mark
7800	unique serial number allocated by PICA

In the record, an at-sign (@) indicates on which word, sorting is to take place.

A field can contain a reference to another field. This is indicated by a dollar sign immediately followed by the tag of the field.

The above KMCs only represent a small number of the total range. Most organizations/libraries only use a selection of KMCs which is relevant to that organization. A PICA record consists of a part with general catalogue entry details that are the same for all organizations (e.g. title, author, imprint), and details that pertain to a particular organization, (e.g. shelf mark, copy details, local classifications and/or subject terms). Agreements are usually made between PICA and the organization, about which KMCs should and should not be used, and which data is stored where and in what form.

Conversion

Because different KMCs are used in different organizations, a routine for importing PICA records into Adlib is always a matter of custom work. Conversion takes place in two phases:

Phase one: downloading PICA GGC to Extended ASCII

The first phase for importing PICA data into Adlib is that of selecting the records in GGC and downloading them to an *Extended ASCII file* consisting of records built up according to the above example. The record header plays an important role in the importing of records in Adlib, because:

1. it contains the Pica Production Number (PPN). The PPN is used in the Adlib import procedure as a relational operator. The Pica Production Number (PPN) is saved in the Adlib database named "catalo" in tag/index "pi". This tag is defined in the Adlib import job as a relational operator. During the import operation, Adlib checks whether the PPN number already exists somewhere in the Adlib database. If so, all data in the record is overwritten with the new values. If not, a new record is added;
2. it serves as a separator between PICA records.

To convert the diacritical characters in the PICA character set (PICA document: DE 007/0195) to Extended ASCII, there must be a conversion file called *user2.txt* on the PICA workstation in the *IBW3*-subdirectory.

An IBW workstation already has a *USER2* download option. This, however, cannot be used for the correct translation of diacritical characters: during downloading, a text file/conversion table must be used, which is based on another file named *picaibm.txt*. The text file (e.g. *picaibm.txt*, *user2.txt*) must be compiled after processing into a PICA program file named *user2.cmp*. This can be done using the PICA command-line program *charconv.exe* as follows:

```
charconv table.txt user2.cmp
```

table.txt	the text file with the conversion table that is based on picaibm.txt
user2.cmp	the PICA-IBW conversion program that translates diacritical PICA characters to Extended ASCII while records are being downloaded from PICA

The text file with the conversion table is divided into two columns. The first column corresponds with the decimal values in the PICA character set, and the second column contains the ASCII values of the diacritical character. A double number in the first column indicates that there are two characters in the GGC: e.g. *225 101 := 138* (meaning: `e, which is to be è. If there is a conversion key in incorrect syntax in the text file, the error message: *macro not defined* will be displayed after compilation with charconv.exe. Conversion keys followed by *NOTDEFINED* are not defined in Extended ASCII.

If all diacritical marks are defined in accordance with the above mentioned method, no further processing will be required before the data in Adlib can be imported. If that is not possible, the program ACCENT.exe can be used to convert the most common diacritics. The syntax of ACCENT.exe is:

```
accent file1.dnl file2.adl
```

file1.dnl	the text file with the PICA download
file2.adl	the converted file to be imported into Adlib

Phase two: uploading PICA records into Adlib.

To upload records in Adlib, you need the following:

- An Adlib import job (e.g. pica.imp), which establishes:
 - General database options;
 - Type of import file: PICA Download;
 - Name of PICA import adapl.
- List of KMCs with corresponding Adlib tags. Such a list may include e.g. the following fields:
 - Name of relational operator (pi).

310#	au
4000	ti

PPN	pi
-----	----

- An import procedure (an ADAPL program) that runs during the importing of the PICA records and processes the contents of the various fields so that they are suitable for entering in Adlib. This import adapl will be custom made for each application because account has to be taken of local modifications. Because of the large number of fields in a PICA download and the time-consuming job of data conversion, such adapls can be quite complex. This requires that the system administrator has a thorough knowledge of programming in ADAPL. The situation is even more complex if the Adlib Loans module is used. In addition to catalogue records, copy records then also have to be created for each separate copy in the PICA download file.

7.5.6 MARC (general ISO 2709)

With Designer, these files can only be imported, not exported. For the tag mapping in an import job, you enter the numbers of the ISO 2709 fields as the source tags, and the Adlib tags as destination tags.

Example of a tag pair definition:

001	A1
008	A2
015	A3
040	A4
041	A5
050	A6
082	A7
090	A8
100	A9
245 \$b \$a \$c \$d	B1

As a rule, each MARC field is placed in one occurrence of the Adlib tag, including subfields. The labels of these subfields are also put in the corresponding field and have to be filtered out during conversion, using an adapl. The subfields of the last field in the above example (tag 245) are put in separate occurrences of Adlib tag B1. The sequence of the occurrences is the same as the order of the subfields

in the tag pair definition.

7.5.7 MARC (Ocelot)

This exchange format was created to allow users of the Ocelot library application to make the transfer to Adlib, and convert their existing databases with this format.

Since then, the need for this conversion has become very rare. So please request detailed information about this subject from our helpdesk, if necessary.

Note that although the name refers to MARC ISO 2709, the format is actually quite different.

7.5.8 MARC (CDS-Isis)

MARC (CDS-Isis) exports data in ISO 2709 format but does not export the two indicator tags for MARC. The existing MARC import in Adlib, removes these indicator tags, and if these are not present, some bytes that contain valid data in CDS-Isis, will unfortunately be removed instead. To avoid this problem, use the separate import format (although comparable to the general MARC format): MARC *CDS-Isis*. This format retains the data at the position where the MARC format stores the indicators.

7.5.9 XML (general, *.xml)

File layout

XML is an excellent format to exchange data with, because it is basically a structured text file, that is readable in Internet Explorer or a text editor, and in the latter you may even edit the file, if you are familiar with the XML language. Also, repeated fields are properly supported: multiple occurrences of fields are typically listed below each other in separate nodes with the same field name (see the example of part of an XML file below). (The XML nodes underneath a record node typically represent the field names.)

```
<adlibXML>
- <recordList>
  - <record>
    <objectname>mug</objectname>
    <objectname>cup</objectname>
```

```

        <objectnumber>087-198</objectnumber>
    </record>
- <record>
    <objectname>dish</objectname>
    <objectnumber>056-012</objectnumber>
</record>
</recordList>
</adlibXML>

```

Conversion

With Designer, these files can only be imported, not exported; from an Adlib application you *can* export to XML though.

For the tag mapping in an import job, you enter the full XML path of the fields in the exchange file as source tags, and the Adlib tags as destination tags.

As the last tag pair you define the record separator: this must be the last (/) node of a record in the XML file.

Example of a tag pair definition:

Source "tag"	Destination tag
<adlibXML><recordList><record><objectname>	OB
<adlibXML><recordList><record><objectnumber>	IN
</record>	<RS>

7.5.10 MS Excel (.xls/.xlsx)

File layout

XLS or XLSX is the default format in which Microsoft Excel stores its files (spreadsheets consisting of columns and rows). These files cannot be imported into Adlib or exported to. However, if you need to import data from an XLS(X) file into Adlib, you can try to convert it to another format first. For example, if your spreadsheet is structured similar to the table below and when field repetitions do not occur, you can easily convert the file to a .csv file, ready to import into Adlib.

Field name 1	Field name 2	Field name 3	Field name 4	etc.
Value 1 (of record 1)	Value 2 (of record 1)	Value 3 (of record 1)	Value 4 (of record 1)	
Value 1 (of record 2)	Value 2 (of record 2)	Value 3 (of record 2)	Value 4 (of record 2)	
Value 1 (of record 3)	Value 2 (of record 3)	Value 3 (of record 3)	Value 4 (of record 3)	
etc.				

Conversion to .csv

1. Open the file in Microsoft Excel.
2. Save the file as a *CSV (MS DOS) (*.csv)* file.
3. Open the file in a text editor to check it. If the first line of your .csv file contains field names, then remove that line because you don't want to import those.
4. The resulting file is ready to import into Adlib. You'll find that double quotes (") in the data have been doubled (") and that data containing double quotes or the field separator (this may be a semi-colon or a comma, depending on your Windows regional settings) has been enclosed in double quotes, just the way Adlib requires the .csv file to be. Empty cells are represented correctly as well. If the file uses a semi-colon as field separator instead of the comma field separator expected by Adlib, you'll have to make sure you define the semi-colon field separator in your import job field mapping explicitly.

7.5.11 Image directory

With Designer, you can import image metadata** into a database through an import job. There are different types of metadata, for instance image metadata such as EXIF and IPTC (contained in photos made with digital cameras), and Windows metadata that is available by default for all types of files and contains the file properties. The Windows metadata is very limited, but may be suitable for images that do not contain any EXIF or IPTC metadata. Prior to application version 4.2 there are no standard Adlib databases and tags for most

tags of any of these types, so you may have to make those yourself with Adlib Designer. Enter at least the following two settings for any of the three indicated types of import (of images) in the import job:

- For the *Input file type* property, choose *Image directory*.
- For the *Source data file* property, you specify the directory in which the images you want to import metadata from are stored. You can also use a wildcard (*) to include all possible values. For example:

C:\MyImages*.jpg	(all jpg files in this folder)
C:\MyImages*.*	(all files in this folder)
C:\MyImages	(all files in this folder)
..\images\obj-1*.jpg	(all jpg files in this folder)

The metadata source tags can now be used in the tag mapping in the import job, depending on whether the image files contain EXIF and/or IPTC metadata.

** Many image files contain not just the image itself, but also so-called metadata: information about the image, like camera settings or descriptions of the photographed subject and the name of the photographer and such. It depends on the way in which an image has been created and edited if, and which, metadata is present. With some software, like Adlib, this metadata can be extracted from an image. Adlib can read EXIF and (from release 6.5.0) IPTC photo metadata. Note that IPTC metadata has no overlap with technical metadata (like EXIF) which is added to a photo automatically by the camera.

EXIF metadata (Exchangeable Image File Format)

Most digital cameras save photos through the EXIF standard; for uncompressed images this means they are saved in the TIFF format, and that compressed images are saved in the JPEG format. Every individual photo file also contains technical metadata about the photo, such as its size, resolution, whether a flash was used, shutter speed, etc.; there may be over a hundred properties for any one photo. This metadata can be extracted through an Adlib import job (or when linking an image to a *Visual documentation* record manually) and loaded into an Adlib database suited for this purpose.

See the EXIF standard definition on the internet for a detailed overview of all EXIF properties (although not all of these can be used in Adlib): <http://www.exif.org/specifications.html> See the table at the

bottom of this Help page for a list of all EXIF properties usable in Adlib.

If the pdf file with the EXIF standard definition cannot be displayed, you may need to download and install software that supports Japanese characters in Adobe Acrobat Reader. You can find this file on:

<http://www.adobe.com/products/acrobat/acrrasianfontpack.html>

Select the font pack for Japanese and the Windows operating system, before starting the download.

You can only use hexadecimal EXIF tags as source tags for the import job you define. (Designer knows that these are hexadecimal tags, so you must not precede such a tag by 0x.) For instance, the hexadecimal EXIF tag 11A that contains the width of the image resolution is entered as the source tag 011A in the tag mapping in a Designer import job. Note that hexadecimal EXIF tags must always be entered as a four-character tag, and that letters in the tag must be written in upper-case.

Additional information and examples of EXIF metadata can be found on:

<http://www.exif.org/>

IPTC metadata (International Press Telecommunications Council)

IPTC is a relatively old metadata standard developed by the International Press Telecommunications Council, applied by Adobe Systems Inc. since the nineties of the last century to describe photos in metadata. So-called IPTC headers may occur in Photoshop, JPEG and TIFF image files. In 2001, Adobe introduced a new metadata framework named XMP (eXtensible Metadata Platform) in her products. The "IPTC Core" XMP schema, developed by the IPTC4XMP initiative since 2004, makes it possible to convert old IPTC headers to the XMP framework and vice versa.

Adlib cannot read XMP metadata, but it does use XMP property and field names to read IPTC metadata – in the specification, the relevant identifier type for contact information is called "field name" and for the other metadata "property name". Via the XMP property/field names you create an implicit mapping to Adlib fields in Adlib. A specification of all XMP property/field names can be found in the "IPTC Core 1.0 specification document" which can be downloaded from <http://www.iptc.org/>.

This metadata can be extracted through an Adlib import job (or when linking an image to a *Visual documentation* record manually) and loaded into an Adlib database suited for this purpose.

Metadata in all Windows files

Every Windows file - irrespective of its type- contains metadata in the shape of file properties. The source fields that you can use in the tag pair mapping for importing are:

- **FILE:** file name;
- **PATH:** full path (including file name);
- **SIZE:** file size;
- **CREATIONDATE:** the ISO date on which the file was first saved;
- **CREATIONTIME:** the time at which the file was first saved;
- **WRITEDATE:** the ISO date on which the file was last modified;
- **WRITETIME:** the time at which the file was last modified;
- **ACCESSDATE:** the ISO date on which the file was last opened;
- **ACCESSTIME:** the time at which the file was last opened.

Setting up your databases and application

EXIF - You could add fields for EXIF metadata to the PHOTO database in Adlib - it makes sense if the description of images in the *Visual documentation* data source in your application would also contain EXIF metadata. From application version 4.2 a number of EXIF fields and an *EXIF data* screen are present by default in this database and data source.

In principle you can name these fields any way you want, but to enable Adlib to automatically merge-in EXIF metadata when the user links an image to a record in *Visual documentation*, these field names have to comply to a certain syntax, namely: `EXIF.PropertyTag<Adlib EXIF field name>`, and have to be the same in all Adlib field name translations. See the table at the bottom of this page for a list of all available Adlib EXIF property field names. To show this metadata in your Adlib application, you'll have to add the new Adlib field tags to existing or new screens.

IPTC - To enable Adlib to automatically merge-in IPTC metadata when the user links an image to a record in *Visual documentation*, the target field names have to comply to a certain syntax, namely: `IPTC.<IPTC property name or field name>`. Replace `<IPTC property name or field name>` by an actual (case-sensitive) property/field name and leave out the brackets. For the "Copyright Notice" metadata for example: `IPTC.CopyrightNotice`, or `IPTC.CiAdrExtadr` for "Contact info: address." Note that the full Adlib field name has to be the same (English) for all interface languages available in your application. Also, if you want all repetitions of any repeated IPTC tags (e.g. for keywords) to be copied to as many occurrences of a repeated Adlib

tag, then the Adlib tag must be a repeated field in the data dictionary as well as on the screen.

To show this metadata in your Adlib application, you'll have to add the new Adlib tags to existing or new screens.

Example

An example of a tag pair mapping including both Windows metadata and EXIF metadata is the following:

Source	Destination
PATH	B1
FILE	FN
WRITEDATE	%d
A002	%x
A003	%y

A002 is the width of the photo, A003 is its height.

Since some information in the source tags may have a format different from what you would like to see in the destination tags, you can write an *adapl* and execute it during import by setting it on the *Options* tab of the import job, as the *Adapl procedure*.

In this example, the metadata will be stored in the *photo* database. The first two destination tags already existed in that database and are defined in the data dictionary, the last three are temporary tags. With an *adapl* we reformat the imported data in B1 and FN and we also reformat and/or concatenate the imported data from the temporary tags and put it in other (existing) *photo* tags. The *adapl* which performs these conversions may look as follows:

```
* images.ada ( import adapl for image metadata, using
* JPEGs with EXIF tags)
* 2005.06.02 Initial coding

* convert URL to relative path
B1 = '..\images' + after$(1, B1, 'images')
FN = cvt$(FN, 2) /* convert file name to lower case
FN = before$(1, FN, '.jpg') /* remove extension

* generate object number
* object number is reproduction ref. without subnumber
IN = before$(1, FN, '-')

* process reproduction date
```

```

BD = right$(%d, 4) + '-' + mid$(%d, 4, 2) + '-' + ~
    left$(%d, 2) /* ISO date

* process reproduction format
* enable the line below if no EXIF tags are available
* FM = FM + ' bytes'

* disable the line below if no EXIF tags are available
FM = %x + ' x ' + %y + ' Pixel'

* default fields
RP = 'JPG' /* reproduction type
ni = 'Conversion Axiell ALM Netherlands' /* input name
di = date$(8) /* input date

* remove temporary fields
%d = null
%x = null
%y = null

end

```

Necessary files

Importing EXIF metadata is available in Adlib from version 5.0. Two new and necessary files for this import functionality are *gdipplus.dll* and *adlibimg.dll*. These files become available automatically after the installation of Adlib 5.0 or higher.

Full list of available Adlib EXIF field names and hexadecimal tags

For more information about the EXIF properties listed below, look up the relevant property in the EXIF standard definition (<http://www.exif.org/specifications.html>) by using the indicated *Hex tag* preceded by 0x: the EXIF field names (the part behind EXIF.PropertyTag) used in Adlib EXIF properties may differ slightly from the official standard EXIF field names.

<i>Adlib field name (case-sensitive)</i>	<i>Hex tag</i>	<i>Comments</i>
EXIF.PropertyTagExifIFD	8769	
EXIF.PropertyTagGpsIFD	8825	
EXIF.PropertyTagNewSubfileType	00FE	
EXIF.PropertyTagSubfileType	00FF	

EXIF.PropertyTagImageWidth	0100	
EXIF.PropertyTagImageHeight	0101	
EXIF.PropertyTagBitsPerSample	0102	
EXIF.PropertyTagCompression	0103	
EXIF.PropertyTagPhotometricInterp	0106	
EXIF.PropertyTagThreshHolding	0107	
EXIF.PropertyTagCellWidth	0108	
EXIF.PropertyTagCellHeight	0109	
EXIF.PropertyTagFillOrder	010A	
EXIF.PropertyTagDocumentName	010D	
EXIF.PropertyTagImageDescription	010E	
EXIF.PropertyTagEquipMake	010F	
EXIF.PropertyTagEquipModel	0110	
EXIF.PropertyTagStripOffsets	0111	
EXIF.PropertyTagOrientation	0112	
EXIF.PropertyTagSamplesPerPixel	0115	
EXIF.PropertyTagRowsPerStrip	0116	
EXIF.PropertyTagStripBytesCount	0117	
EXIF.PropertyTagMinSampleValue	0118	
EXIF.PropertyTagMaxSampleValue	0119	
EXIF.PropertyTagXResolution	011A	Image resolution in width direction
EXIF.PropertyTagYResolution	011B	Image resolution in height direction
EXIF.PropertyTagPlanarConfig	011C	Image data arrangement
EXIF.PropertyTagPageName	011D	
EXIF.PropertyTagXPosition	011E	
EXIF.PropertyTagYPosition	011F	
EXIF.PropertyTagFreeOffset	0120	

EXIF.PropertyTagFreeByteCounts	0121	
EXIF.PropertyTagGrayResponseUnit	0122	
EXIF.PropertyTagGrayResponseCurve	0123	
EXIF.PropertyTagT4Option	0124	
EXIF.PropertyTagT6Option	0125	
EXIF.PropertyTagResolutionUnit	0128	Unit of X and Y resolution
EXIF.PropertyTagPageNumber	0129	
EXIF.PropertyTagTransferFunction	012D	
EXIF.PropertyTagSoftwareUsed	0131	
EXIF.PropertyTagDateTime	0132	
EXIF.PropertyTagArtist	013B	
EXIF.PropertyTagHostComputer	013C	
EXIF.PropertyTagPredictor	013D	
EXIF.PropertyTagWhitePoint	013E	
EXIF.PropertyTagPrimaryChromaticities	013F	
EXIF.PropertyTagColorMap	0140	
EXIF.PropertyTagHalftoneHints	0141	
EXIF.PropertyTagTileWidth	0142	
EXIF.PropertyTagTileLength	0143	
EXIF.PropertyTagTileOffset	0144	
EXIF.PropertyTagTileByteCounts	0145	
EXIF.PropertyTagInkSet	014C	
EXIF.PropertyTagInkNames	014D	
EXIF.PropertyTagNumberOfInks	014E	
EXIF.PropertyTagDotRange	0150	
EXIF.PropertyTagTargetPrinter	0151	
EXIF.PropertyTagExtraSamples	0152	
EXIF.PropertyTagSampleFormat	0153	
EXIF.PropertyTagSMinSampleValue	0154	
EXIF.PropertyTagSMaxSampleValue	0155	
EXIF.PropertyTagTransferRange	0156	

EXIF.PropertyTagJPEGPProc	0200	
EXIF.PropertyTagJPEGInterFormat	0201	
EXIF.PropertyTagJPEGInterLength	0202	
EXIF.PropertyTagJPEGRestartInterval	0203	
EXIF.PropertyTagJPEGLosslessPredictors	0205	
EXIF.PropertyTagJPEGPointTransforms	0206	
EXIF.PropertyTagJPEGQTables	0207	
EXIF.PropertyTagJPEGDCTables	0208	
EXIF.PropertyTagJPEGACTables	0209	
EXIF.PropertyTagYCbCrCoefficients	0211	
EXIF.PropertyTagYCbCrSubsampling	0212	
EXIF.PropertyTagYCbCrPositioning	0213	
EXIF.PropertyTagREFBlackWhite	0214	
EXIF.PropertyTagICCProfile	8773	This TAG is defined by ICC for embedded ICC in TIFF
EXIF.PropertyTagGamma	0301	
EXIF.PropertyTagICCProfileDescriptor	0302	
EXIF.PropertyTagSRGBRenderingIntent	0303	
EXIF.PropertyTagImageTitle	0320	
EXIF.PropertyTagCopyright	8298	
// Extra tags (Like Adobe Image Information tags etc.)		
EXIF.PropertyTagResolutionXUnit	5001	
EXIF.PropertyTagResolutionYUnit	5002	
EXIF.PropertyTagResolutionXLengthUnit	5003	
EXIF.PropertyTagResolutionYLengthUnit	5004	
EXIF.PropertyTagPrintFlags	5005	
EXIF.PropertyTagPrintFlagsVersion	5006	
EXIF.PropertyTagPrintFlagsCrop	5007	
EXIF.PropertyTagPrintFlagsBleedWidth	5008	

EXIF.PropertyTagPrintFlagsBleedWidthScale	5009	
EXIF.PropertyTagHalftoneLPI	500A	
EXIF.PropertyTagHalftoneLPIUnit	500B	
EXIF.PropertyTagHalftoneDegree	500C	
EXIF.PropertyTagHalftoneShape	500D	
EXIF.PropertyTagHalftoneMisc	500E	
EXIF.PropertyTagHalftoneScreen	500F	
EXIF.PropertyTagJPEGQuality	5010	
EXIF.PropertyTagGridSize	5011	
EXIF.PropertyTagThumbnailFormat	5012	1 = JPEG, 0 = RAW RGB
EXIF.PropertyTagThumbnailWidth	5013	
EXIF.PropertyTagThumbnailHeight	5014	
EXIF.PropertyTagThumbnailColorDepth	5015	
EXIF.PropertyTagThumbnailPlanes	5016	
EXIF.PropertyTagThumbnailRawBytes	5017	
EXIF.PropertyTagThumbnailSize	5018	
EXIF.PropertyTagThumbnailCompressedSize	5019	
EXIF.PropertyTagColorTransferFunction	501A	
EXIF.PropertyTagThumbnailData	501B	RAW thumbnail bits in JPEG format or RGB format, depends on EXIF.PropertyTagThumbnailFormat
// Thumbnail related tags		
EXIF.PropertyTagThumbnailImageWidth	5020	Thumbnail width
EXIF.PropertyTagThumbnailImageHeight	5021	Thumbnail height

EXIF.PropertyTagThumbnailBitsPerSample	5022	Number of bits per component
EXIF.PropertyTagThumbnailCompression	5023	Compression scheme
EXIF.PropertyTagThumbnailPhotometricInterp	5024	Pixel composition
EXIF.PropertyTagThumbnailImageDescription	5025	Image tile
EXIF.PropertyTagThumbnailEquipMake	5026	Manufacturer of image input equipment
EXIF.PropertyTagThumbnailEquipModel	5027	Model of image input equipment
EXIF.PropertyTagThumbnailStripOffsets	5028	Image data location
EXIF.PropertyTagThumbnailOrientation	5029	Orientation of image
EXIF.PropertyTagThumbnailSamplesPerPixel	502A	Number of components
EXIF.PropertyTagThumbnailRowsPerStrip	502B	Number of rows per strip
EXIF.PropertyTagThumbnailStripBytesCount	502C	Bytes per compressed strip
EXIF.PropertyTagThumbnailResolutionX	502D	Resolution in width direction
EXIF.PropertyTagThumbnailResolutionY	502E	Resolution in height direction
EXIF.PropertyTagThumbnailPlanarConfig	502F	Image data arrangement

EXIF.PropertyTagThumbnailResolutionUnit	5030	Unit of X and Y resolution
EXIF.PropertyTagThumbnailTransferFunction	5031	Transfer function
EXIF.PropertyTagThumbnailSoftwareUsed	5032	Software used
EXIF.PropertyTagThumbnailDateTime	5033	File change date and time
EXIF.PropertyTagThumbnailArtist	5034	Person who created the image
EXIF.PropertyTagThumbnailWhitePoint	5035	White point chromaticity
EXIF.PropertyTagThumbnailPrimaryChromaticities	5036	Chromaticities of primaries
EXIF.PropertyTagThumbnailYCbCrCoefficients	5037	Color space transformation coefficients
EXIF.PropertyTagThumbnailYCbCrSubsampling	5038	Sub-sampling ratio of Y to C
EXIF.PropertyTagThumbnailYCbCrPositioning	5039	Y and C position
EXIF.PropertyTagThumbnailRefBlackWhite	503A	Pair of black and white reference values
EXIF.PropertyTagThumbnailCopyRight	503B	Copyright holder
EXIF.PropertyTagLuminanceTable	5090	
EXIF.PropertyTagChrominanceTable	5091	
EXIF.PropertyTagFrameDelay	5100	

EXIF.PropertyTagLoopCount	5101	
EXIF.PropertyTagPixelUnit	5110	Unit specifier for pixel/unit
EXIF.PropertyTagPixelPerUnitX	5111	Pixels per unit in X
EXIF.PropertyTagPixelPerUnitY	5112	Pixels per unit in Y
EXIF.PropertyTagPaletteHistogram	5113	Palette histogram
// EXIF specific tags		
EXIF.PropertyTagExifExposureTime	829A	
EXIF.PropertyTagExifFNumber	829D	
EXIF.PropertyTagExifExposureProg	8822	
EXIF.PropertyTagExifSpectralSense	8824	
EXIF.PropertyTagExifISOSpeed	8827	
EXIF.PropertyTagExifOECF	8828	
EXIF.PropertyTagExifVer	9000	
EXIF.PropertyTagExifDTOrig	9003	Date & time of original
EXIF.PropertyTagExifDTDigitized	9004	Date & time of digital data generation
EXIF.PropertyTagExifCompConfig	9101	
EXIF.PropertyTagExifCompBPP	9102	
EXIF.PropertyTagExifShutterSpeed	9201	
EXIF.PropertyTagExifAperture	9202	
EXIF.PropertyTagExifBrightness	9203	
EXIF.PropertyTagExifExposureBias	9204	
EXIF.PropertyTagExifMaxAperture	9205	
EXIF.PropertyTagExifSubjectDist	9206	
EXIF.PropertyTagExifMeteringMode	9207	
EXIF.PropertyTagExifLightSource	9208	

EXIF.PropertyTagExifFlash	9209	
EXIF.PropertyTagExifFocalLength	920A	
EXIF.PropertyTagExifMakerNote	927C	
EXIF.PropertyTagExifUserComment	9286	
EXIF.PropertyTagExifDTSubsec	9290	Date & time sub-seconds
EXIF.PropertyTagExifDTOrigSS	9291	Date & time original sub-seconds
EXIF.PropertyTagExifDTDigSS	9292	Date & time digitized sub-seconds
EXIF.PropertyTagExifFPXVer	A000	
EXIF.PropertyTagExifColorSpace	A001	
EXIF.PropertyTagExifPixXDim	A002	
EXIF.PropertyTagExifPixYDim	A003	
EXIF.PropertyTagExifRelatedWav	A004	Related sound file
EXIF.PropertyTagExifInterop	A005	
EXIF.PropertyTagExifFlashEnergy	A20B	
EXIF.PropertyTagExifSpatialFR	A20C	Spatial frequency response
EXIF.PropertyTagExifFocalXRes	A20E	Focal plane X resolution
EXIF.PropertyTagExifFocalYRes	A20F	Focal plane Y resolution
EXIF.PropertyTagExifFocalResUnit	A210	Focal plane resolution unit
EXIF.PropertyTagExifSubjectLoc	A214	

EXIF.PropertyTagExifExposureIndex	A215	
EXIF.PropertyTagExifSensingMethod	A217	
EXIF.PropertyTagExifFileSource	A300	
EXIF.PropertyTagExifSceneType	A301	
EXIF.PropertyTagExifCfaPattern	A302	
EXIF.PropertyTagGpsVer	0000	
EXIF.PropertyTagGpsLatitudeRef	0001	
EXIF.PropertyTagGpsLatitude	0002	
EXIF.PropertyTagGpsLongitudeRef	0003	
EXIF.PropertyTagGpsLongitude	0004	
EXIF.PropertyTagGpsAltitudeRef	0005	
EXIF.PropertyTagGpsAltitude	0006	
EXIF.PropertyTagGpsGpsTime	0007	
EXIF.PropertyTagGpsGpsSatellites	0008	
EXIF.PropertyTagGpsGpsStatus	0009	
EXIF.PropertyTagGpsGpsMeasureMode	000A	
EXIF.PropertyTagGpsGpsDop	000B	Measurement precision
EXIF.PropertyTagGpsSpeedRef	000C	
EXIF.PropertyTagGpsSpeed	000D	
EXIF.PropertyTagGpsTrackRef	000E	
EXIF.PropertyTagGpsTrack	000F	
EXIF.PropertyTagGpsImgDirRef	0010	
EXIF.PropertyTagGpsImgDir	0011	
EXIF.PropertyTagGpsMapDatum	0012	
EXIF.PropertyTagGpsDestLatRef	0013	
EXIF.PropertyTagGpsDestLat	0014	
EXIF.PropertyTagGpsDestLongRef	0015	
EXIF.PropertyTagGpsDestLong	0016	
EXIF.PropertyTagGpsDestBearRef	0017	
EXIF.PropertyTagGpsDestBear	0018	
EXIF.PropertyTagGpsDestDistRef	0019	

7.5.12 Modes

With Designer (from version 6.1.2054), Modes exchange files can be imported (not exported).

Modes type exchange files are hierarchical, for example:

```
*RECORD_NUMBER NMIAA : 1234.1004
  *IDENTIFICATION
    *NAMED_COLLECTION Ceramics Collection
    *SIMPLE_NAME tile
    *BRIEF_DESCRIPTION Ecclesiastical tiles and fragments of
tiles (1897.1100 - 1137), from Christchurch Cathedral (in
Antiquities Register).
    *AUTHORITY3 T2
    *NUMBER_OF_ITEMS 1
    *DESCRIPTION
      *HEIGHT 2.8cm
      *LENGTH 12cm
      *WIDTH 12cm
      *CONDITION f
    *PHOTOGRAPHY
      *PHOTOGRAPH_NUMBER NMIRP : 112.13
    *PERMANENT_LOCATION KS.BC & 26.2 : 29.10.1996
    *RECORDER CLO : 29.10.1996
    *ACQUISITION
      *METHOD purchase
      *PERSON : Frazer, Dr.
    *ASSOCIATION
      *PLACE & Christchurch Cathedral & Dublin & Co. Dublin &
Ireland
#
```

```
*RECORD_NUMBER NMIAA : 1234.1005
  *IDENTIFICATION
    *NAMED_COLLECTION Ceramics Collection
    *SIMPLE_NAME tile
    *BRIEF_DESCRIPTION Ecclesiastical tiles and fragments of
tiles (1897.1100 - 1137), from Christchurch Cathedral (in
Antiquities Register).
    *AUTHORITY3 T2
```

```

*NUMBER_OF_ITEMS 1
*DESCRIPTION
  *HEIGHT 2.9cm
  *LENGTH 11cm
  *WIDTH 11cm
  *CONDITION f
*PHOTOGRAPHY
  *PHOTOGRAPH_NUMBER NMIRP : 112.14
*PERMANENT_LOCATION KS.BC & 26.2 : 29.10.1996
*RECORDER CLO : 29.10.1996
*ACQUISITION
  *METHOD purchase
  *PERSON : Frazer, Dr.
*ASSOCIATION
  *PLACE & Christchurch Cathedral & Dublin & Co. Dublin &
Ireland
#

```

A field name is indicated by a preceding *, and # is the default record separator. The indentation level of a field name indicates its hierarchical level. Further, a record may begin with *****record*****, if so, this line will not be imported by Adlib.

For the tag mapping in an import job, you must enter the full hierarchical path of a Modes field as the source tag, and the Adlib tags as destination tags.

Example of a tag pairs definition:

*RECORD_NUMBER*IDENTIFICATION*NAMED_COLLECTION	A1
*RECORD_NUMBER*IDENTIFICATION*SIMPLE_NAME	A2
*RECORD_NUMBER*BRIEF_DESCRIPTION	A3
*RECORD_NUMBER*DESCRIPTION*HEIGHT	A4
*RECORD_NUMBER*DESCRIPTION*LENGTH	A5

7.5.13 Other formats/separators

The fields and records in exchange files are separated by specific characters dependent on the exchange format. If you want to import an exchange file that is just like one of the formats that Designer can import, but has different separator characters, you can just change the separators for that format.

If you have a file structured like for instance an ASCII delimited one, but in which the field separators are not commas but some other character, you can still import it as an ASCII delimited file; then it's necessary though to specify the separator used in this file. In the tag mapping in an import job, you include `<FS>` as a destination tag. For the source tag you provide either the hexadecimal ASCII value of the separator character, or just the character itself when it is printable.

Example of a tag pair in which a TAB (a non-printable character) is specified as separator:

0x09	<FS>
------	------

(Hexadecimal values start with 0x.)

Example of a tag pair in which a semicolon is specified as separator:

;	<FS>
---	------

You can also specify a record separator in this way, but now use `<RS>` as destination tag. In Adlib tagged files for instance, the default record separator is `**`, but when you have a similar file to import, that separates records with say, only one `*`, you can specify this in the tag mapping:

*	<RS>
---	------

8 General topics

§

8.1 Adlib file types and folders

Depending on your license type, you have more or fewer modules. Some of the subfolders mentioned underneath will probably not be on your system. Said subfolders are valid from application version 2.2.0. From 2.1.0 those folder names have been changed gradually. In the *Contents* column of the table below, the most used older folder names are still mentioned.

Subfolder	Contents
<i>adapls</i>	Contains all ADAPL source text files with the <i>.ada</i> extension (previously located in the <i>\adapl sources</i> folder), and all compiled adapls with the <i>.bin</i> extension (previously located in the <i>\adaplbina</i> and <i>\data</i> folders, and print adapls in the application folders).
<i>data</i>	Contains all databases (<i>.cbf</i> files), database structures (<i>.inf</i> files), index files (<i>.00#</i>), <i>.cnt</i> files that are used for automatically numbered fields, and subfolders in which pointer files are saved.
<i>executables</i>	Contains Adlib executables and DLLs, with their own system texts (<i>.txt</i>) and Help files (<i>.hlp/.adh</i>), and also your licence file and a <i>\documentation</i> subfolder with in it the release-notes. Previously these files were subdivided into the main Adlib programs (in a <i>\bin</i> folder) and the old tools ADSETUP, DBSETUP, ACSETUP and ADAPL (in a <i>\tools</i> folder).
<i>images</i>	Contains your images.
<i>Library</i> and/or <i>Museum</i> application folders	Contains application and/or module definitions (<i>.pbk</i> files), an Adlib application logo, and <i>DOS4GW.exe</i> that should be in every application folder. In <i>\library circulation management</i> you'll also find templates (<i>.rtf</i>) for issue-slips and the like, that are used in that application.
<i>screens</i>	All screen files that are used in your Adlib applications, are located in this folder. Previously,

	application specific screens resided in the application folders, zoomscreens in <code>\data\zoom</code> , and link screens in <code>\links</code> .
<i>texts</i>	Contains one text file per language, for all print adapls together.
<i>worddoc</i>	Contains per subfolder the letters you generate with Word templates.
<i>worddoc\templates</i>	Contains Word templates in different languages, installed by default.

File types in Adlib

You will find several types of files in Adlib folders. Some of these are inherent to Adlib, other types you may also encounter elsewhere on your system. To be able to modify the Adlib applications and/or database, for installing software upgrades or for making backups and to limit the risk of losing data, it is useful to know what kind of information is stored in each type of file. Underneath is an alphabetized list of all types of files in an Adlib application.

000, 001, etc.: index files.

Every database has its own index files, stored in the subfolder *data*. An index contains the values of a key field from all the records, which refer to those same records, and is used by Adlib to quickly search through the database. The 000-file contains all record numbers (primary reference: priref) followed by a memory address for the location of that record inside the database (the *.cbf* file). From 001 the indexes are defined in and can be changed in Adlib Designer in the database setup. The content of index files is generated automatically when a database is filled.

ADA: ADAPL source files.

These text files contain the program code for external bits of programs that for instance check user input. These source files can be modified, and new source files can be made in any text editor. The compiled version of these source files can be used by Adlib if this is specified in the database or in the application.

ADH: Help files.

The Help topics that are available everywhere in an Adlib-application are stored in *.hlp* or *.adh* files. The older *.hlp* extension is being phased out in favour of the *.adh* extension that was introduced in

model application 4.2: however, besides the extension, there are no differences between the two file types.

There should be a help file for each available interface language (0 = English, 1 = Dutch, 2 = French, and 3 = German, amongst others). These files can be viewed or modified with a rich-text editor, like Wordpad. There are *applic#.hlp/.adh* files and *adlib#.hlp/.adh* files; only *applic#.hlp/.adh* files may be modified by the user.

BIN: compiled adapls.

Compiled ada-files get the extension *.bin*.

BMP: image files (bitmaps).

The Adlib-logo for instance, is stored in a bmp-file.

CBF: database files.

A *.cbf* file holds the complete contents of all records from one database. You may for instance find *COLLECT.cbf*, *EXHIBIT.cbf*, *THESAUS.cbf* and *PEOPLE.cbf*. The records are stored sequentially, so that indexes are necessary to find the correct record quickly.

DAT: exported cbf-files, stored as ASCII text files.

When you export an Adlib database, the target file should have the extension *.dat*. Such a file is a snapshot of a database, and will no longer contain the same information as the *.cbf* file after it has been modified.

DLL: Dynamic Link Library-files.

A collection of small programs that are only opened by a larger program when they are necessary, to save memory space.

ERR: error report.

When errors occurred during import, export or recovery through pre-Designer tools, they were registered in an err-file in the *\data* folder. This file could be viewed with a text editor.

In Designer, any errors will be reported in the main window of the toolkit.

EXE: executables.

Exe-files contain executable codes. *Adlwin.exe* for instance, is the Adlib software necessary to search through a database and to enter new data. The user interface from which you access the software is called the application; which application you are using depends on

the *Start in* folder where the software searches for the application files.

EXP: export job.

You can define export jobs in Adlib Designer. Such files get the extension *.exp*.

FMT: screen files (a.k.a. forms, or tabs).

Every screen is defined in its own *.fmt* file. Entry fields and other screen elements can be associated with database fields, in order to represent (part of) the content of a record from that database.

HLP: Help files.

The Help topics that are available everywhere in an Adlib-application are stored in *.hlp* or *.adh* files. The older *.hlp* extension is being phased out in favour of the *.adh* extension that was introduced in model application 4.2: however, besides the extension, there are no differences between the two file types.

There should be a help file for each available interface language (0 = English, 1 = Dutch, 2 = French, and 3 = German, amongst others). These files can be viewed or modified with a rich-text editor, like Wordpad. There are *applic#.hlp/.adh* files and *adlib#.hlp/.adh* files; only *applic#.hlp/.adh* files may be modified by the user.

IDX: free text index.

Wordlist.idx contains every word from all free-text (word) indexed fields from all Adlib databases. Every word is assigned its own unique number (in sequence of input). There is a separate index (see **000, 001, etc.**) for all free-text indexed fields (such as *Title* and *Description*) which only contains word numbers from *wordlst.idx* and a reference to the records in which the referred words for these word numbers appear.

IMP: import job.

You can define import jobs in Adlib Designer. Such files get the extension *.imp*. An import job is necessary for importing data during e.g. conversions.

INC: general ADAPL source files.

ADAPL source code that can be used in several adapls can be put into a separate source file (an *.inc* file) that can be included during a compilation of an *.ada* file.

INF: information files.

These files store the structure of a database (without the actual data): which datasets there are (which types, length, and which tags), whether there are linked fields and, if any, to which databases and index they are linked. This structure is defined in the database setup in Adlib Designer.

ISU: de-installation data.

The changes made to the system by an Adlib installation, such as placing program files and shortcuts, and changes in the Windows-registry are coded and saved in an isu-file. Windows uses this file if you want to de-install Adlib.

Please remember that when you de-install Adlib all previously entered data will be lost! For an upgrade of your installation with a service release or completely new version, you do not need to de-install Adlib first.

Only those files that do not hold data, namely executables, system texts and system help texts, are automatically replaced during the installation of an upgrade; data, customized applications and settings are saved. De-installation can be done from the Windows Control Panel (click the *Software* icon).

LCK: record lock files.

When you change a record in Adlib, it is locked so that it is impossible for two persons to work on the same record at the same time. For Adlib CBF databases, this information is temporarily stored in a locks file belonging to the database (for an Adlib SQL database, the record locks are stored in their own *...recordlocks* SQL table). The record lock is removed from the *.lck* file (or from the SQL table) when you stop editing the record. When no records are being edited anywhere on your system or network, the *.lck* files for all databases (or the relevant SQL table) should be empty.

Once an *.lck* file has been created, it is not automatically removed once it becomes empty. This was designed that way, because constantly making and removing *.lck* files takes time.

When you can't edit certain records whilst you're absolutely sure that nobody currently is editing records but you, and your access rights indeed permit you to edit these records, there might be a corrupted *.lck* file on your system for the current database (or obsolete record locks are present in the relevant SQL table). (An *.lck* file might become corrupted after a crash of your software while editing a record.)

You can manage all record locks via the *Record lock manager* tool, present in Adlib Designer or available separately, to remove faulty

record locks. (See also: Managing record locks.)

LIC: licence file

You own an *adlib.lic* licence file, specific for your contract with Adlib. To be able to use the Adlib software, this file (or copies of it) and the associated *adliblic.dll*, must be present in all directories that contain Adlib *.exe* files, normally these are the *\executables* or *\bin* (and possibly *\tools*) subfolders. (See the separate chapter in the installation guide for more information about licences.)

LST: listing of an inf-file.

In Designer and in the old Adlib DBSETUP tool you can produce documentation for a database, which offers a complete overview of the database structure. For this, DBSETUP created a text file (in the subfolder *\data*) with the extension *.lst*. An *.lst* file is always a snapshot. In Designer you can save the documentation as an XML file.

OUT: print files.

When you print to a file instead of to a printer, this file is given the standard extension *.out*. The file can be opened in a text editor.

PBK: application definition files (parameter blocks).

A *.pbk* file registers which files (databases) are accessible in an Adlib application, and inside that which indexes, which screens and which output formats can be used. When Adlib starts, the *.pbk* file indicated in the desktop shortcut is used to start a particular application. If you start Adlib without selecting a *.pbk* file, Adlib does not know which application you want to start, and you are prompted to select a parameter file (a *.pbk* file).

ROLES files.

A *roles* text file (without extension) contains a list of roles names. This file was used by the old tools DBSETUP and ADSETUP to offer a list of predetermined roles when you were assigning users to roles or access rights to roles. Adlib Designer does not use this file.

TXT: (system) text files.

These files contain system messages and labels for buttons. In the file name, the number represents the language in which the file has been written (English = 0, Dutch = 1, French = 2, German = 3, Arabic = 4, Italian = 5 and Greek = 6). Text file names without language extension cannot be used: even a text file in the default language

(English) should have the "0" language extension. In the past, text files without a language extension were possible as well, yet only when they were stored in a language specific folder, pointed to by the ADLIB_DIRx variable, where x indicated the language number; this practice is now deprecated.

(Also see Creating Help for Adlib applications.)

XML: various files

XML stands for eXtensible Markup Language. Files in this format are text files that structure their content using certain tags. This format is designed to be customized and therefore so flexible that Adlib will use it more and more to structure and store it's settings and data. You should not change the contents of such files manually.

8.2 Searching for Adlib objects

You can search your Adlib system for the names of either adapls, datasets, fields, roles, screens, tags, help keys, or domains, that appear in any of four possible file types: *.pbk* (application definitions), *.fmt* (screens), *.inf* (data structures), and *.bin* (compiled adapls: here you do not search in the adapl source code, but just in the title of adapls). For such a search, you must use the *Object searcher* tool in Designer.

With the search result, you can edit your application in the *Application browser*, without the risk over overlooking any reference.

Start this tool by choosing *Tools > Object searcher* or by clicking the button for it in the main *Adlib Designer* window :



Execute a search

First select the desired work folder by clicking the *Folder* button in this tool. Then select the *Object category* you are searching for, and type the name of that object (without extension) in the *Search for* entry field. Under *File specification*, select the file types that you want to search in, for the object name, and mark any of the *Options* to be able to search:

- independent of upper or lower case (*Ignore case*);
- on a partial term and not only on full names (*Search substrings*);
- including subfolders of the currently selected work folder (*Include subfolders*);

- and remove the results of any previous search from memory (*Clear previous results*).

Finally, click the *Start* button in this tool to begin searching. You may cancel a search in progress by clicking the *Stop* button.

The search result

The search result will be displayed in the main *Designer* window. You can save this result as an *.rtf* text file (to be able to view it at any later time from a text editor), by choosing *File > Save result window as*, or by clicking the *Save as* button:



Non-modality of the Object searcher

The *Object searcher* stays active (and running if the search is still in progress) when you close it, which means that the search result is still being written to the main *Designer* window, and that if you open the tool again you'll see the results of the current/last search.

You can also open other tools after closing the *Object searcher* while a search has not finished yet, and switch back and forth from the main *Designer* window with the search result, to the opened tool, when you implement changes in your application while keeping the search result at hand as a list of things to do.

8.3 Backups, and logging and recovery

In case something goes wrong with your computer, network, or software, it is very important that you do not lose your Adlib application and all the data in your Adlib databases.

It is only a small effort to protect this data, so that when necessary you can retrieve it up to the last record that was added or modified. You can do this through backups and through Logging & recovery.

Requirement 1: regularly back up your data.

A backup is a (sometimes compressed, e.g. by zip) copy of your data. Depending on the frequency with which your data is changed, we advise you to make a backup of your data once a day, once a week or once a month. Always make a backup before you start reindexing or importing, or when you intend to make changes to the application or restart a logging file.

CBF - In case your applications run on the Adlib proprietary CBF database platform, you can make a backup by copying Adlib files to e.g. a CD-ROM, a ZIP-drive or a tape. (If necessary, you can compress the data for storage, so that it will take up less space.) It is sensible to copy **all** Adlib-files (the entire Adlib directory, not just the databases), so you can be sure that you are securing all your data. If you do not use Adlib Designer (or the no longer distributed ADSETUP and DBSETUP), and your application has never been changed, it will suffice to copy the (*data* subfolder.)

SQL - In the case of an Adlib SQL Server or Adlib Oracle database, you'll have to use the relevant server management software to create backups of the database. Changes in your application can be secured by copying your main Adlib folder to e.g. a CD-ROM, a ZIP-drive or a tape. A database backup could be created as follows:

1. In Microsoft SQL Server Management Studio Express, right-click the database, and in the pop-up menu choose *Tasks > Back up*.
2. Set the *Backup type* to *Full*. This will save the entire database, including transaction log, in a back-up.
3. Set other options (also see the *Options* tab) as desired.
4. Click *OK* to generate a backup.

It is also important to make backups of your digital images. The Adlib database normally only holds a reference to a digital image, the image itself should also be kept safe; when you store the images in an *Images* folder inside the Adlib directory, the images will automatically be backed-up during a complete *Adlib* directory backup.

For all backups goes: test the backups that you make from time to time, and save them separately somewhere, away from your original files.

You can test a backup of Adlib (using *cbf* databases) by placing the (uncompressed) backup Adlib directory somewhere in a temporary directory on your computer, making shortcuts to this directory like you did for your main system, and simply starting and trying out applications. Delete the temporary folder and the temporary shortcuts after testing, to prevent confusion.

Requirement 2: keep an automatic logging file.

On loss of application information, you will have your application(s) back by copying and pasting a backup. This applies to databases as well, in case your applications run on the Adlib proprietary CBF database platform, and you wish to restore a corrupt database. However, records that were added or modified after that last backup,

will not be retrieved by this safety measure alone. To solve this problem – to make sure that you can retrieve even the latest modified record without having to make backups continuously – you must use Adlib logging & recovery (automatic registration and retrieval of modifications). After placing the databases back, you then complete a further restore procedure on the basis of the Adlib logging file, to bring the databases up-to-date.

In the case of a corrupt SQL/Oracle database, the database must be restored via the relevant server management software: this will automatically bring the backup up-to-date with the aid of the transaction log which has been kept by the server.

The Adlib software can take care of logging and recovery, regardless of the database platform on which your Adlib applications run, but SQL Server and Oracle have their own logging & recovery system which offers advantages over that of Adlib. You will mostly want to use Adlib's logging & recovery for *cbf* databases; for Adlib SQL and Adlib Oracle databases you can switch it off and use the platform's proprietary procedures.

Adlib logging - For Adlib's own logging & recovery, logging means that all the records that are added to the database, changed or removed, are stored in an extra logging file: in this case, a logging file is an Adlib-tagged file in ASCII format. (So changes in applications or in database structures are not registered.)

You can *set* this automatic registering per *cbf* database in Designer. For every database definition, enter the same name for the logging file, e.g. `F:\security\log`. In this file, Adlib registers for every record that is changed from which file it came, so that multiple databases can be logged in one logging file.

A *logging file* name may be preceded by a drive letter; it is also sensible to save a logging file to another disk than the one that holds Adlib, so that if the Adlib disk were to become inaccessible, the logging file would still be usable. Do save the logging file on a disk that is accessible while you are working with Adlib, so that every change can be registered.

To ensure a correct recovery, it is essential that all databases refer to the same logging file. You will then be able to start a recovery for any database, automatically restoring any other databases as well. (If you were to use a different logging file for every database, you would have to restore all these databases separately and in a particular order.)

SQL logging – In Adlib for SQL, the basic operations which comprise write actions, are bundled in so-called transactions. The SQL recovery model makes sure that transactions are saved only if all basic operations of which they consist, have been completed successfully. Microsoft SQL Server supports three different recovery models:

- *Simple*, the default and most basic recovery model, in which each successful transaction is removed immediately. Therefore, the log remains very small. This won't allow you to restore a database to an earlier point in time.
- *Bulk logged*, in which all transactions will be logged, except bulk operations. (In this context, Adlib import and export are no bulk operations.)
- *Full*, in which all transactions will be logged. Adlib advises to use this model.

The recovery model could be set as follows:

1. In Microsoft SQL Server Management Studio Express, right-click the database, and choose *Properties* in the pop-up menu.
2. Select the *Options* tab and set the *Recovery model* to *Full*.
3. Open the *Files* tab and set a path to the logging file: behind the log-ging file in the *Database files* list, click the ... button (just left of the *Path* column). Make sure that the logging file (and backups) are not stored on the same disk as the database itself.

Adlib recovery (repair with Adlib's proprietary logging file)

With Adlib logging, you save changes in your data from the moment you set it. When a file has become corrupted, and you wish to repair your Adlib system, first put back a functioning backup. You can restore the changes made after the backup, by reading and executing the logging file through the *Recovery* tool in Adlib Designer.

To start this tool, choose *View > Recovery* in the main *Designer* menu or click the button for it:



1. First make sure you have selected the right work folder (see the status bar of the tool window). This should be the folder in which you have just placed a backup file to recover, to make it your new live Adlib application. If necessary, choose *File > Working folder* or click the button for it, to select another folder.
2. On the *General* tab for the *Recovery file* option, you select the logging file you need to import. For *Meta data tag* you can provide a tag in your Adlib system in which you want to import the metadata that is present in the logging file (this is information about dates of modification, and so on). The meta data tag is not mandatory.

3. On the *Run* tab you can start your recovery, if you have selected a logging file. Statistics about the recovery are presented in real-time.

Every action in the logging file that was executed before you made the backup that you are now restoring, is automatically executed too, even though these changes are unnecessary. In principle this shouldn't cause any errors, but it is recommended to rename the logging file in Windows Explorer each time you make a backup (add for instance the date); this causes Adlib to create a new logging file automatically (each time with the same name, because you didn't change that name in the database setup). This way you always have the smallest logging file available when you need it for recovery, that has been kept up-to-date since the last backup; this can save you a lot of time during recovery, and prevents possible errors due to redundant recovery changes.

Should you want to do a recovery from a date prior to the moment of renaming the old logging file (e.g. because the backup you want to use is older too), then you will have to run a recovery twice (or as much times as there are logging files up to that date). In the first recovery, you enter for the import file the name you gave to the old logging file when you renamed it. Then you run a second recovery, and use the current logging file as an import file. The repair is now completed from the backup you put back up, up to and including the last change that was made to your data. With multiple logging files you first recover the oldest logging file (from the date of the backup), then the second oldest, and so on, up to and including the most recent logging file.

So, reading a logging file and executing the data modifications is done with the *Recovery* tool in Designer. When errors occur during a recovery, the error report is displayed in the main Designer window - it is no longer saved to a file with the extension *.err*, and the name of the logging file. It is important to check for any errors after a recovery.

Recovery should always be tried first in case of a database corruption. To establish whether you have a database corruption, you can call our help desk. And again: it is important to use a backup that was made at a time when all appeared to be functioning normally (this is not necessarily the last backup, it can also be an earlier one).

NB In principle, recovery is meant to repeat actions from the past, not to undo them. But if you wish to use recovery to retrieve records that were removed accidentally, you can change history by making some changes in the logging file (with a text editor): every deleted file is marked with a '-' (minus sign) in the logging file. By removing these

minus signs before a recovery, you are fooling the program and the records are placed back.

SQL Server database restore (repair with .ldf log)

A corrupt Adlib SQL Server database can be repaired with the *Restore* function. By default, the most recent backup of the backup and the active transaction logging file are used to bring the database up-to-date again, but you may restore to an earlier time as well.

Before you can start restoring the database, you'll have to make a backup of the transaction log, for instance as follows:

1. In Microsoft SQL Server Management Studio Express, right-click the database, and in the pop-up menu choose *Tasks > Back up*.
2. Set the *Backup type* to *Transaction log*. This will only save the transaction log in a back-up.
3. On the *Options* tab, mark the *Back up the tail of the log...* option. Set other options as desired
4. Click *OK* to generate the backup.

After that, you could use *Restore* as follows:

1. In Microsoft SQL Server Management Studio Express, right-click the database you wish to repair, and in the pop-up menu choose *Tasks > Restore > Database*.
2. In *From database*, select the backup which should be used, or, in *From device*, select the logical or physical backup device (for instance a tape drive or files) which contains the backup.
3. In the *Select the backup sets to restore* list, mark the components of the backup which you want to use for restoring (the recovery plan suggested by default - the already marked components - is probably advisable).
4. On the *Options* tab you may set other options. The *Help* function on each tab offers information about all functions.
5. Click *OK* to execute the restore procedure.

Managing the logging file

Adlib logging file - In principle, an expanding logging file (mostly used for *cbf* databases) is no problem. Should this file get too large though (more than 600 MB), you can decide to close the current file by renaming it (e.g. with Windows Explorer); you could use today's

date in the file name, for instance. However, do not change the name of the logging file in the database setup! Adlib will automatically create a new logging file with the old name, which will register the changes made from that moment on. The old, now renamed logging file can then be stored somewhere else, if you wish, on a CD-ROM for example.

SQL Server logging file – The logging file of an Adlib SQL Server data-base keeps on growing if you use the *Recovery model: Full* (as advised by Adlib). If the transaction log is getting too big, you'll have to purge it. The best way of doing that is by creating a *Full* backup of your SQL database first (via the server management software), then setting the *Recovery model* to *Simple*, followed by reducing the transaction log via the *Shrink* option, after which you reset the *Recovery model* to *Full*, for example as follows:

1. In Microsoft SQL Server Management Studio Express, right-click the database, and in the pop-up menu choose *Tasks > Back up*. Set the *Backup type* to *Full* and set other options as desired, to generate a backup.
2. Right-click the database, and in the pop-up menu choose *Properties > Options*. Set the *Recovery model* to *Simple*.
3. Right-click the database, and in the pop-up menu choose *Tasks > Shrink > Files*. Set the *File type* to *Log*. Check the *Available free space*, select *Reorganize pages...* as the *Shrink* action, and set an accompanying file size (e.g. 1 MB) before you click *OK* to shrink the logging file.
4. Right-click the database, and in the pop-up menu choose *Properties > Options*. Set the *Recovery model* to *Full*.

Securing changes in applications

Modifications in your applications cannot be registered in a logging file. But every time you make a backup of the entire Adlib directory, you also backup any changes you have made to the application yourself. So ensure that you make a backup before you make any changes to the application, and make another backup afterwards. You can use the first backup in case the new application does not work properly, and the second backup if the new application disappears from your system or gets corrupted. You can also retrieve separate screens (*.fmt* files) from a backup.

Note that Adlib logging & recovery only works well if the backup you are using contains exactly the same databases for which the logging file has registered changes. This means: if you have made any changes to a database structure since you ran the backup you now

intend to use, then recovery with the logging file will cause problems, because it may refer to database fields that were not present in the database structure of the backup. So please be careful with this.

8.4 User authentication and access rights

Adlib offers application user login functionality, and a comprehensive access rights mechanism to define precisely which users have what rights when they're working with Adlib. This functionality extends the standard Windows single sign-on authentication, which of course should always be the first line of defence.

User authentication through application user login

From Adlib 6.0 you have the possibility to set up an application-bound login procedure. Then, when starting each (instance of an) application for which this has been set, first a login window appears in which the user must enter his or her user name, password and possibly an (Active Directory) domain. No access will be granted to the application until the user fills in the correct information.



You, or your system administrator, has to set up this way of authentication explicitly. As long as that has not been done, Adlib will use the user credentials from the Windows login for applying any security policy.

Make this setting on the *Application authentication* tab of the properties of an application structure file (.pbk) that you have opened in the *Application browser* in Designer. So, per application you can set

this user authentication.

First choose one of four possible methods for saving the user names and passwords:

- **Adlib.pbk** - If you've already assigned roles to users in your *.pbk* file, to be able to apply access rights to the different parts of your application and/or databases, then it's a small step to assign passwords to all those users. These user names and passwords will then be necessary to log in, from now on.
If you select this method, you needn't fill in any other properties on the *Authentication* tab.
Just select each user in an application structure in the *Application browser* in Designer, and fill in a *Password* of your choice for each. (Note that you have to register really all users of this application in the *pbk*.)
- **Adlib database** - You may also save all user names, passwords and (optionally) roles in one of your databases, for instance your *Persons and institutions* (PEOPLE) database. Although then you'll probably have to create three new fields in the data dictionary first (for the user name, the password and the role of the user) and also place those fields on a screen, so that you'll be able to enter and save a user name, password and role in existing records with personal data. Moreover, you'll have to create an index for the user name field. For each user a record must then exist, or must be created, that at least holds information in the user name and password fields.
When that's done, you can set the authentication method *Adlib database* for an application. Then, on the same properties tab, choose the *Database* to which you have added the fields, by clicking the ... button. The path will then be filled in automatically in the *Folder* property. If necessary you can also choose the proper dataset. And finally you must select the data dictionary fields that you've created for the user name, password and possibly role, in the *User id field*, *Password field* and *User role field* options (use the ... buttons next to them).
If you have more than one Adlib application and certain users must have different roles in different applications, then you'll have to add a fourth field to your user data, namely an application id field. Every application must have been assigned its own unique application id which can be referenced in your application id field. By grouping your new user role and application id fields in the datadictionary and making this group repeatable on the screen, you allow for the possibility of each user having a different role in every application. On the *Application authentication* tab you must now set the *Application Id field* you

implemented.

- **Active Directory** - Active Directory is the basis for distributed networks on Windows systems, and offers a secure and structured way to store data about so-called objects in a network. One such an object is a user. In Active Directory, a system administrator can create accounts for a/o all users of resources in a Windows 2000 domain, in which user name and password are registered. When starting an Adlib application, Windows checks the login information through these user accounts, and grants permission to use the application, if appropriate.
- **HTTP** - This authentication method allows you to verify a user's credentials by sending a URL to a web server over HTTP or HTTPS. The web server is expected to respond with the standard HTTP Response Code *200 OK* if the validation succeeds; any other response code is interpreted as a failed authentication. You must define the exact form of the URL that will be sent to the web server by providing a format string in the (authentication) Format string property of the relevant application (click the link for more information about the format syntax). Once the user has been authenticated, Adlib will use the user name to look up the role for the user, and the Adlib security policy through access rights will be applied.
(HTTP authentication is available from Adlib 6.1.0.)

Save the edited *pbk* file after you have set an authentication method, and (close and) start the concerning application to test your settings. Note that, whatever way you choose to have users log in, the name with which the user logs in will be used to assign access rights through the roles mechanism (if that is used in your application). So make sure that any user names linked to roles, match the user names as registered in the authentication model that you chose.

The Adlib access rights mechanism

In the Adlib access rights mechanism screen files, objects in a database definition (namely databases, datasets and fields), and objects in an application definition (namely the application, data sources, methods, export jobs, output jobs and friendly databases), can all be protected. (Methods are access points and software functionality like searching in different ways, creating new records, deleting records, using pointer files, global updating, importing, exporting, and deriving.)

In Adlib Designer this can be set up in the following ways, that may be combined:

- Use the roles functionality.
- Adjust the standard security for an application.
- Use the authorisation functionality.
- Use Windows NT groups in defining roles.

The user credentials required for this mechanism to work, are obtained from the Windows user authentication if no Adlib application user authentication has been applied, otherwise the latter provides the necessary credentials.

Use the roles functionality

In an application definition you can include a list of all users of the application. In the properties of each user you then assign a so-called role to which the user belongs (such as `staff`, `user`, `public`, etc.). Then you may include any role on the *Access rights* properties tab of an Adlib file or object (as listed in the introduction above) and assign access rights to the role. Access rights that are applied this way, apply not only to the file or object itself but also to all sub-objects in it, which saves you work. And access rights that you apply to roles for objects in an application definition can only be further restricting than any access rights applied to the same roles for objects in a database definition; they can never be extended.

One special role is `$REST`. This role should never be assigned to users explicitly. Its purpose is to be able to specify access rights to the current object for all users who's role has not been assigned access rights to this object and for all users without a role. So if you've defined users and roles in the pbk, but for an object you do not assign access rights to these roles, while you do specify that `$REST` should get e.g. read access, then all users will have only read access. However, by default `$REST` has not been set anywhere, and if you do not assign access rights to any roles, then all users will have full access.

A second special role is `&ADMIN` (available from Adlib 6.6.0). This role should be assigned to administrator users, but access rights must never be applied explicitly to this role because users who get the `$ADMIN` role have full access rights by default, plus the right to unlock manually locked fields and even the right to change the name of the record owner and user access rights per record if record authorisation is active for the database. The `$ADMIN` role and its inherent rights also have priority over access rights assigned to `$REST`. The `$ADMIN` role and its rights even overrule access rights for application roles. The latter means that users with the `$ADMIN` role may get to see an Adlib application in quite a different way, namely with all possible details

screens, methods, access points and all possible data sources: this is because in model application 4.2 and higher, application roles are the filter that determines if an object appears in a certain application or not. Since that filter is ignored for an \$ADMIN user, he or she gets to see all application objects.

Note that on opening of the *Application browser*, it only loads Adlib objects from the currently set root folder in memory; any objects in a subfolder will only be loaded too, if you open that folder. The advantage of this is that the start-up time of the *Application browser* is being reduced. Other than that, it is only relevant for your work when you are editing access rights for application or database objects: in the *Roles* drop-down lists that you encounter on those properties tabs you'll find all harvested roles from all currently loaded Adlib objects (so you won't have to remember them or look them up all the time). But to allow Designer to harvest all roles from database and application objects, you must have opened the data and application folder(s) nodes at least once in this session, and they will remain in memory when you close those nodes again.

Access rights and roles, for methods - If you are about to apply access rights to roles for methods, you'll find there are many combinations of method type and access rights of which the consequences may not be immediately clear. Therefore, the following table provides an overview of all possible combinations and their consequences.

For instance, if you set a method of the *Delete records* type (because you want at least some users to be able to remove records from the current data source), and you have specified a number of users that fall into two roles, *staff* or *public*, you may assign the *public* role any of the access rights *None*, *Read* or *Write* because they have the same effect that records cannot be deleted. Because *staff* must have *Full* access rights for this method, you don't need to assign these: Full access rights are applied to any role for a method by default (unless the application's default access rights have been restricted), if a role's access rights have not been set for this method.

Everything in the table in light green is staying on the safe side, disabling functionality or different types or searching; everything in cyan means enabling some or all functionality of that method.

Access rights	<i>None</i>	<i>Read</i>	<i>Write</i>	<i>Full</i>

Method type				
<i>Term search</i>	Searching is not allowed.	Searching is allowed.	Searching is allowed.	Searching is allowed.
<i>Free text search</i>	Searching is not allowed.	Searching is allowed.	Searching is allowed.	Searching is allowed.
<i>Create new records</i>	Creating a new record in the current data source is not possible.	Creating a new record in the current data source is not possible.	Creating a new record in the current data source is allowed.	Creating a new record in the current data source is allowed.
<i>Expert search</i>	Searching with the search language is not possible.	Searching with the search language is allowed.	Searching with the search language is allowed.	Searching with the search language is allowed.
<i>Query by Form</i>	Searching with a QBF is not possible.	Searching with a QBF is allowed.	Searching with a QBF is allowed.	Searching with a QBF is allowed.
<i>Delete records</i>	Records in the current data source cannot be deleted.	Records in the current data source cannot be deleted.	Records in the current data source cannot be deleted.	Records may be deleted from the current data source.
<i>Pointer files</i>	No access to pointer files.	Pointer files can be read, but not written out, nor deleted.	Pointer files can be read and written out, but not deleted.	Pointer files can be read, written out and deleted.
<i>Global Update</i>	Global update is not allowed.	Global update is not allowed.	Global update is permitted.	Global update is permitted.
<i>Print records</i>	Records cannot be printed.	Printing is permitted.	Printing is permitted.	Printing is permitted.
<i>Export records</i>	Records from the current data source cannot be exported.	Exporting is allowed.	Exporting is allowed. Export jobs may be saved but	Exporting is allowed. Export jobs may be saved and

			not deleted.	deleted.
<i>Import records</i>	Importing records into the current data source is not possible.	Importing records into the current data source is not possible.	Importing records into the current data source is permitted. Import jobs may be saved but not deleted.	Importing records into the current data source is permitted. Import jobs may be saved and deleted.
<i>Derive records</i>	Deriving records into the current data source is not permitted.	Deriving records into the current data source is not permitted.	Deriving is permitted, but the original record cannot be deleted.	Deriving is permitted and the original record may be deleted.
<i>Date range search</i>	Searching on a date range is not permitted.	Searching on a date range is allowed.	Searching on a date range is allowed.	Searching on a date range is allowed.
<i>Link update</i>	Disables the Thesaurus update functionality.	Disables the Thesaurus update functionality.	Enables the Thesaurus update functionality.	Enables the Thesaurus update functionality.
<i>Fixed query</i>	Searching with a fixed query is not permitted.	Searching with a fixed query is allowed.	Searching with a fixed query is allowed.	Searching with a fixed query is allowed.
<i>Merge terms</i>	Disables the Merge terms functionality.	Disables the Merge terms functionality.	Enables the Merge terms functionality.	Enables the Merge terms functionality.
<i>Named range search</i>	Searching on a named date range is not permitted.	Searching on a named date range is allowed.	Searching on a named date range is allowed.	Searching on a named date range is allowed.
<i>Location change procedure</i>	Changing the current location of a marked set of objects all at once is not	Changing the current location of a marked set of objects all at once is	Changing the current location of a marked set of objects all at once is	Changing the current location of a marked set of objects all at once is

	possible.	not possible.	allowed.	allowed.
<i>Print barcodes</i>	Single-click printing of labels to a label printer is not possible.	Single-click printing of labels to a label printer is allowed.	Single-click printing of labels to a label printer is allowed.	Single-click printing of labels to a label printer is allowed.
<i>Publish records to The Collection Cloud</i>	Uploading or deleting records to and from The Collection Cloud is not possible.	Uploading or deleting records to and from The Collection Cloud is not possible.	Uploading or deleting records to and from The Collection Cloud is allowed.	Uploading or deleting records to and from The Collection Cloud is allowed.

The application *Identification* role - A different type of role is the application *Identification* property. You do not assign users to this role, because this role automatically applies to everyone using that application! With this role, a standard screen for instance, may be read-only in one application, whilst it may be editable in another application.

More details - See the Help topics for the *Access rights* tab in the properties of a specific Adlib object, for detailed information about access rights for that particular object.

Adjust the standard security for an application

In the *Default access rights* property of an application definition, you may set the standard access rights for this application, that must apply if no access rights have been set for an object through the roles functionality. If a role and access rights have been mapped for any application object or database object, then those access rights always have priority over the *Default access rights* for the application (whether those role-specific access rights allow more or fewer user actions than this default).

Use the authorisation functionality

With this method of security you can set access rights per record per user or role. In each record you then list users and/or roles* that have different access rights from the norm. This is useful if some records in a database may only be accessed by some employees because of sensitive contents for example, while other records may be accessed freely.

There are two different ways of implementing this functionality:

- The first is a general way with which you can either:
 - completely *Exclude* the users or roles listed in the record and hide this record from them, while to all other users any other access rights settings apply;
 - or instead allow (*Include*) the users or roles listed in the record (and those users and the record owner only) full access to this record while excluding all other users implicitly.This has to be set for the database as a whole, so you can't decide per record that you now want to enter an exclude list, and for another record an include list because that would be the least amount of work in both cases. If one of both mentioned access rights extremities is all you need for a database, then use the *Exclude* or *Include* method, since this is quicker to implement and easier to use.
- The second way (available from Adlib 6.1.0, and Designer 6.1.1871) is more specific and eliminates the limitations of the first implementation. This has taken shape in the third *Authorisation type* called *Specified rights*. Choose this type if you want to be able to indicate per record which users or roles have which specific access rights (*None, Read, Write* or *Full*). With the *Specified rights* method, the initial creator of a record (a user or role), the so-called *Record owner*, always has full access to that record, and in this record only the creator is allowed to set and change which users have which access rights, or transfer the record to another record owner. The latter is necessary for instance when the current record owner is a user and this user changes employment; using a role as record owner instead (possible from 6.6.0) circumvents this problem because if records are property of a role, an application manager with access to Adlib Designer can always assign that role to other users so that they also get the right to adjust the access rights of others to the relevant records.

* You can only use a user name in this functionality if no role has been assigned to that user in the application definition (whether that role is actually used anywhere else or not). This is because when Adlib searches for access rights to apply, it searches by means of the role assigned to a user. So if a user James has the role Management and you apply the authorisation functionality discussed here, you will have to enter the Management role in records instead of user name James. Of course, then the record access rights apply to all users with the Management role, not just to James. Users without roles however, can indeed be entered in records; you can mix user names and roles here.

All applicable database settings can be found in the *Authorisation and*

security box on the *Database properties* tab for a selected database in the *Application browser*. So you decide per database which authorisation method you want to use (or none at all of course).

- Implementing general authorisation through Exclude or Include

1. Firstly, in the data dictionary of the desired database you must define a new text field that can hold user names and roles.
2. Secondly, an entry field associated with this database field, must be placed on one or more screens that are used in the current application to access said database.
3. In the properties of this database you can specify the *Authorisation user field* that you defined in step 1, and its *type* (*Exclude* or *Include*). When creating or editing each record, the user must supply the names of all users or roles to which he or she wants to apply the *Exclude* or *Include* authorisation that you set here. (Under NOVELL, you can also use group names, instead of user names.) *Exclude* is analogous to *None* access rights; *Include* is analogous to *Full* access rights. You choose the type on the basis of the smallest number of user names or roles that would apply to it. For instance, if there are just a few people that you want to exclude from some records, you choose the *Exclude* type. For the appropriate records you only have to provide the names or the roles of these few people each time. Users that are excluded through authorization from seeing certain records cannot acquire *Read*, *Write* or *Full* access rights through any of the other security functionality in Adlib.

- Implementing specific authorisation through Rights

1. Open the drop-down list *Authorisation type* and choose the *Specified rights* method.
2. The *Default access rights* concern the access rights for all users to a record in which those users or roles have not been explicitly assigned access rights. If you are more likely to list users or roles in a record, that have extensive access rights like writing or also being allowed to delete, then for this option you'll probably want to set limited default access rights, like read-only or no access at all. The reverse situation will probably occur less often. But it is of course important to choose these default rights carefully, because you won't be listing the majority of users or roles in each new record.
3. For the other three options you set three different field names. The *Authorisation (user) field* already existed before 6.1.0 and may already be filled in; if so, then leave it as it is, if not, then fill in a new field name or tag. Later, in this field in the running application, the user names or roles will be filled in of users to

whom the creator of the record wants to assign specific rights. (User names are dependent on the authentication method that your application uses.)

4. For the *Authorisation rights tag*, fill in a new field name or tag. In this field, per user or role the specific access rights will be stored that are assigned by the creator of a record.
5. For the *Record owner tag*, also fill in a new field name or tag. Later, in this field the user name or role of the creator of a record will be stored automatically. This name or role is determined through the login of the current user.
6. For Record owner type, choose between *Current user* (the creator of a record will be its owner) and *Current role* (the role of the creator of a record will be its owner).
7. For the three or two new fields you'll also have to make data dictionary definitions. Create these new fields in the field list of the current database. All three must be normal text fields that you may give a limited length because these fields only need to contain names or access rights. The fields are not linked or enumerative: the access rights field for instance, will be provided with an access rights drop-down list automatically, you don't need to specify this yourself. What is important though, is that the user/role field and the access rights field must be repeatable and that these fields must be defined in a data dictionary field group together!
8. Now the database side is finished. Next, you'll have to create a new screen or adjust an existing screen, on which these fields must be filled in later on. Let's assume that you adjust an existing screen. *Management details* for instance, would be quite suitable for this purpose. Open the desired screen in the *Screen editor* in Designer. If the user field is not yet present on the screen, then insert three screen fields (otherwise two) with accompanying labels, that you associate with the data dictionary fields that you created in the previous steps. You can leave all screen field properties to their default values.

In the running application, only the owner of a record (a single user or all users with a certain role) can fill in or change the three new fields. All other users can never adjust the contents of these three fields (unless there is no owner at all). If they have access to the record, then at the most they can read these fields, even if they have full access.

Each time that a user now tries to open and/or edit a record in this database, Adlib will check if that user has the appropriate access rights.

Use Active Directory groups in defining roles

The Active Directory (or Windows NT) or group to which a user belongs, can be used to specify the access rights for an Adlib object. This way you only need to register the individual users in the appropriate Active Directory groups and then register those Active Directory groups in the Adlib application definition. Simply include the group name in the *Users* list of the relevant Adlib application definition and assign it an appropriate Adlib role. For use within an *adlwin.exe* application only, you can name such a role anyway you like, but for *wwwopac.ashx* (the Adlib API) to be able to apply access rights to such users, the role name must be identical to the Active Directory group/Adlib user name because the API has no access to the application definition containing the Adlib user-to-role mapping and can only use the Active Directory group name of the logged-in user as a possible Adlib role (access rights in Adlib are always assigned to roles). So we recommend that when you include an Active Directory group name as an Adlib user in the Adlib application definition, you assign it an Adlib role with exactly the same name.

The Adlib security system will then not only check the Adlib access rights of the Adlib role for the specific Active Directory group (specified as Adlib user), but also the Windows access rights for the Active Directory group under which the current Windows user falls. The most limited access rights of the two will be used if more than one restriction is used.

For example, the Active Directory group *students* may have *Write* access (on Windows/SQL level) to a specific Adlib SQL database, but "John", who is a member of that group may have only *Read* access for a specific field in Adlib. In this case (for safety reasons) the *Read* access has priority as far as the relevant field is concerned. In this example, an Adlib user named *students* with the role *students* has been specified in the Adlib application definition and the relevant data dictionary field has been assigned *Read* access rights for the role *students*.

If a user is linked to several NT or Active Directory groups, the individual rights are assigned to the group in which the user has the most rights. Individual rights always take precedence over group rights.

Temporary Adlib tags that are not specified in the data dictionary always have full rights, independent of surrounding rights. This prevents side-effects in such temporary fields, caused when *Read* access rights are assigned to the databases, on Windows level or from Adlib. If said temporary field has no write authorization, no value can be assigned to it by the software itself: a value must be written to a database, even though it is temporary.

8.5 Status management of authority records

From Adlib 6.5.0, terms and names forced from within a catalogue into an authority database like the *Thesaurus* or *Persons and institutions* by clicking the *Add term* button in the *Linked record search screen*, are assigned the "candidate" status if in the relevant database a field has been set up for it. In that case, a *Show forced terms* checkbox will appear in the *Linked record search screen*. Only when you mark this checkbox, forced terms or names will be added to the displayed list of found keys. With an unmarked checkbox, the list of keys shows all non-forced terms or names: these are terms or names which have been created as new records from within the *Thesaurus* or *Persons and institutions*, or as a new record from within a zoom screen to the *Thesaurus* or *Persons and institutions*.

To be able to really do something with this functionality, application and database adjustments are necessary: by default, no data dictionary field has been set up in the model applications up to and including 3.5, to store this status in; however, in newer model applications, status management is implemented fully. You can also adjust your own application yourself if you have Adlib Designer. Then proceed as follows:

1. You may create a status field in any linked database. However, you should only do this in databases in which you would like to set and maintain the status of entered terms, for instance because spelling rules have to be checked or guidelines for thesaurus terms have to be observed. The *Thesaurus* and *Persons and institutions* come to mind first, but the principle is valid for any database. Open the *Application browser* in Adlib Designer and create in e.g. the *Thesaurus* (the THESAU file) a new enumerative field with the name `term.status` and the tag `ts` (if that name and tag do not exist yet, that is). On the *Enumeration values properties* tab, you must specify the following static list:

Value	Language text (English)
0	undefined
1	approved preferred term
2	approved non preferred term
3	candidate
4	obsolete
5	rejected

Note that the status of a term can have one of six different values. Terms forced into the *Thesaurus* from within another database, automatically receive the value 3 "candidate". For every language on the *Default values* tab you can, if you wish, enter the number of the value which you would like to see filled in in the status field by default when you create a new record in the *Thesaurus*: if you mainly enter approved preferred terms for example, then 1 would be appropriate, but if you equally often enter candidate terms, approved preferred terms and approved non-preferred terms then 0 would probably be the best default value. Let's look at the different values for a moment: 0 means that no status has been assigned yet, 1 must be assigned to approved preferred terms, 2 to approved non-preferred terms, 3 to candidate terms - forced terms are candidate terms automatically, 4 to obsolete terms which probably may still be used, and 5 to rejected terms. The advantage of assigning status 5 instead of removing a rejected term altogether is that the information about rejecting the term is saved: no-one will ever attempt to enter an earlier rejected term as a preferred term again, simply because it is still present in the database as a rejected term.

2. In the properties of the relevant authority database (in our example the *Thesaurus*) on the *Advanced* tab, you'll find the new *Thesaurus term status field* option. Enter here the name or tag of your new status field, and save the changes in this database.
3. Now you must place the field on a screen for the authority database. For the *Thesaurus* you can do this on *thes.fmt* (*Thesaurus term*), below *Term code* for instance - this screen may have a different name in your application though. The new screen field must be of data type *Text*, *Not repeatable*, and have *Read and write* access rights.
Remember to save your changes in the screen file.

You are done as far as the *Thesaurus* is concerned. In the running application you can now assign a status to every term in this database. Everywhere in other databases where a field links to the *Thesaurus* you may include candidate terms in the list of found keys in the *Linked record search screen*, or leave them out. And when you force terms into the *Thesaurus*, they automatically receive the candidate status.

Setting access rights for showing candidate terms

The *Show forced terms* checkbox can be hidden from certain user groups via access rights, if you don't want everyone to be able to

view and/or link candidate terms. Set those access rights in the *Access to view candidate terms in link window* list on the *Advanced* tab of the relevant authority database. The checkbox will be hidden from users with a role which has been assigned the access rights *No*. By default, users do have access to candidate terms.

8.6 Managing record locks

When a user starts editing a record, Adlib applies a so-called record lock. This prevents colleagues from editing the same record simultaneously: as long as there is a lock on a record, others can only view that record, not edit it. If another user still tries to edit a locked record, Adlib will prohibit this. Adlib will only release a record when the user editing the record leaves edit mode by saving or closing the record unsaved.

In normal operation, Adlib cleans up a record lock as soon as a user stops editing the record, but if during editing of a record a malfunction in your system occurs (e.g. because of a crash), it is possible that the record lock is not removed, which makes it impossible for anyone to edit that record again (because it is still locked).

If you can open a record in display mode, but you cannot edit or remove it while no-one else is editing the record and you have write or full access to it, this is probably because the record is still locked. You may receive an error 34 or 85, and/or a message stating that the record is in use or that no record lock can be applied. The record lock must first be removed before you can edit or delete the record.

In such cases, you can use the *Record lock manager* to check which records are still locked, and remove the record locks that are no longer applicable. This functionality is a standard part of Adlib Designer but is also freely available as a separate tool to all Adlib customers so that users of Adlib Basis or Adlib Lite can manage record locks as well.

The Record lock manager

With the *Record lock manager* tool you can view and remove record locks. Start the separate tool (if you have it) by double-clicking *AdlibRecordLocks.exe* in the `\AdlibRecordLocks` folder of your Adlib system, or start it from Adlib Designer by choosing *Tools > Record lock manager* in the menu or by clicking the button for it in the main *Adlib Designer* window :



1. First select the `\data` subfolder of your Adlib folder as the working folder for this tool.



The *Record lock manager* then provides a real-time overview of all current locks in all databases in that folder: any record opened or closed in your Adlib applications is immediately reflected in the locks overview.

2. The date a lock was created, the record number, and by whom, are displayed in the overview to help you determine if a lock is obsolete. And locks that are older than 15 minutes turn red. If the list is long, you may want to sort it by clicking the appropriate column header.

In the status bar of this tool window the most recent "sample" time is displayed (the time at which this tool actively searched for locks; but whenever a lock is created, this data is immediately transferred to the record lock manager and displayed, so the list is updated in real-time.

3. Before removing any faulty locks you may want to document all currently present locks. Click the *View a record lock list* button to generate this documentation.



You can print this documentation and/or save it as an XML file.

4. Finally, select a lock, or more than one by **Shift-** or **Ctrl-**clicking, and remove them by clicking the *Delete selected record lock(s)* button, but only if you are sure that no-one is currently editing those records!



You may also delete all record locks at once by choosing *Edit > Delete all locks* or by clicking the button for it.



See also

Adlib file types and folders

8.7 Colour your Adlib application

From Adlib 6.0 you can use 32-bit colours for the design of a/o screens and the elements on them. You have the following possibilities:

Application background colour

You can set the background colour of a whole application at once. This concerns the colour of the window pane in which the *Search wizard* appears, for instance, and the tab sheets of a detailed display. You will only see that colour when a small window is displayed in that larger window pane (*Search wizard*, *Pointer files* and *Expert search system*); when tab sheets are being displayed, the background of said window pane is not visible.

A background colour can be set on the *Application properties* tab of a selected application structure (.pbk file) in the *Application browser* of Adlib Designer. The option is called: *Application background colour*. Click the coloured box next to it to open a standard Windows colour picker. You can select a basic colour, or click *Define Custom Colours* to choose another colour. Save the adjusted .pbk file and (close and) open the concerning Adlib application to view the result.

Screen background colour

Per screen (tab sheet) you may set a background colour. This is the colour that will be visible around the fields and their labels (boxes are currently always transparent).

Open a screen that you wish to change, in the *Screen editor* of Designer. Right-click the screen, but not in a box or other screen element, and choose *Properties* in the pop-up menu that opens. The screen properties will open.

At the bottom of the first properties tab you'll see the *Screen background colour* option. Click the coloured box next to it to open a standard Windows colour picker. Choose a colour and close the properties dialog, save the changes to the screen and (close and) open the concerning Adlib application to admire the result in the detailed or brief display for which this screen is intended.

Colours for all labels and fields on a screen

Per screen (tab sheet) you can set foreground and background colours for all labels and fields on it*, all at once.

Open a screen that you want to change, in the *Screen editor* of Designer. Right-click the screen, but not in a box or other screen element, and choose *Properties* in the pop-up menu that opens. The screen properties will open. At the bottom of the first properties tab

you'll see the *Foreground colour* and *Background colour* options for *Labels*, *Data* (fields and their contents) and *Highlighted data* (a selected record on this screen for brief display - so this doesn't apply to detail screens).

The foreground colour is always the text colour, while the background colour concerns the colour behind that text. Click the coloured box that you want to set differently, to open a standard Windows colour picker. You can select a basic colour, or click *Define Custom Colours* to choose another colour. Close the properties dialog, save the changes to the screen and (close and) open the concerning Adlib application to view the result in the detailed or brief display that uses this screen.

* Note that the colours that you have assigned to individual labels or fields (see next paragraph) won't be overwritten if you assign colours to all labels and fields at once, like described above.

Colours for an individual label or field

It's also possible to apply a colour to individual labels and/or fields. This way you can stress the importance of certain fields, for instance. Colours that you've set for individual labels and fields, will normally not be overwritten by colours that you set on any other level. Open a screen that you wish to change, in the *Screen editor* of Designer. Right-click a field or label and choose *Properties* in the pop-up menu that opens. The properties of that screen element will be opened. At the bottom of the first properties tab you'll see the *Foreground colour* and *Background colour* options (for a field, or for a label) and an example next to it. Click the coloured box that you want to set differently, to open a standard Windows colour picker from which you can choose a new colour. Close the properties dialog, save the changes to the screen and (close and) open the concerning Adlib application to view the result in the detailed or brief display for which this screen is intended.

Applying colour schemes

To be able to set the same colours for all labels and fields on some screens or all screens in an entire application at once, you'll have to work with colour schemes. So you may apply a colour scheme to a single screen or to a whole application:

- per screen

Open the properties of a screen that you edit in the *Screen editor*, and in it click the *Colour schemes* button to edit and/or apply a scheme to the current screen only.

By default, there are seven colour schemes listed in the *Adlib colour scheme editor* window. Click one of those schemes to preview the

screen with this scheme applied to it, and to view the definition of that scheme, in the example tab sheet in the bottom half of this window, and through the coloured boxes beneath it.

If you like one of the offered schemes, then double-click it to select it and close this window. The colours of this scheme are applied to the screen. Note that colours that were assigned to individual fields earlier, will not be overwritten.

By the way, the *Standard Adlib* scheme is the colour scheme that you were accustomed to in Adlib applications.

You can also create new colour schemes. Click the *Create new colour scheme* button in this window to add a new scheme.



Click the "*New colour scheme*" name in the list with schemes twice (do not double-click), and type a new name for that scheme. Then select the desired colour for each of the different screen elements, by clicking the coloured boxes at the bottom of the window.

If you are done creating the scheme, click the *Save image file* button to store your current collection of colour schemes. This collection is saved in an XML file (*ColorSchemes.xml*) in your Adlib Designer folder. If you'd like to create a new colour scheme on the basis of another colour scheme, you simply select the existing colour scheme, click the *Copy* button and then *Paste*, to add a copy to the list, of which you can then change the name and colours.

A redundant colour scheme may be removed by selecting it and clicking the *Delete colour scheme* button.

Always save your collection after you have edited a scheme or removed it, to store the changes in said XML file.

Note that the *Tools* menu in this window offers the possibility for the currently selected colour scheme to copy a label colour to the screen colour (or vice versa), because you might want both background colours to match.

- to an entire application

You may apply a colour scheme to an entire application too, or rather: to all screens in a folder that you select.



To this end, click the *Change your application colours* button in the main window of Adlib Designer (or choose *Tools > Application colouring tool* in the menu of that window). The *Application colour change tool*

window opens.

In this window, first choose a folder that holds the screens to which you want to apply a colour scheme, by clicking the *Folder* button. If your application uses screens from different folders (for instance from *\screens*, *\data\zoom* and your application folder), then process them all at once, by just selecting your Adlib Software folder of which these three folders are subfolders. Then mark the *Include subfolders* option in the *Application colour change tool* window. This way, all screens that are used in your Adlib application(s), will be adjusted automatically later on.

Now you may pick a colour scheme through the *Colour schemes* button (then double-click a scheme to select it; *Alice blue* complements the Adlib 7 colour scheme), or choose a few colours manually through the coloured boxes in the upper part of this window.

If you would like colours set for individual fields, labels, system fields and boxes on screens to be overwritten by the colour scheme that you execute here, then also mark the *Change all fields* option.

Finally, click the *Apply* button to apply the chosen colours to the selected folder(s). The *Progress* bar continuously displays the percentage of processed files.

Close this window when the procedure has finished, click the *Save all* button in the main window of Designer to save the changes to all screens, and (close and) open the concerning Adlib application to view the result.

8.8 Windows Image Acquisition

From 6.1, support for Windows Image Acquisition has been added to Adlib. With this functionality you can link images to records in Adlib, images that you create at that same moment and save in a desired format in a folder of your choice, with a device for making still images that is connected to your computer: these are devices like scanners, digital photo camera's, web cams or digital video camera's. In other words: when you fill in a visual documentation record, you can scan an image or make a picture from within Adlib, save this image any way you choose, and link it to the current record immediately. And all this in a few simple steps. A prerequisite is that the relevant device* supports WIA (you can check this in the documentation of the device), and that you have a Windows XP or higher system.

Preparing your application for WIA

To be able to read in images with WIA when you create a visual documentation record, it is necessary to make a few settings for the image reference field (usually *B1* in version 3.4 or older applications, and *FN* in version 3.5 or newer applications) in your visual documentation database (usually *photo*), through Adlib Designer. Only for fields of the data type *Image* you can make these settings: a properties tab is present for such a selected data dictionary field in the *Application browser*, called *Image properties*.

1. First choose a *File name assignment type*. You can choose between *From field content* or *Automatic sequential generation*:
 - *From field content* means that the user has to enter a new file name manually for the image that will be read in subsequently, when generating an image to be linked for an image reference field, via the *From scanner or camera...* button in Adlib. (See the Adlib user guide, for button icons used in Adlib.)
 - *Automatic sequential generation* means that the user reads in a new image through the *From scanner or camera...* button, without having to enter a file name for it, because that name will be generated automatically via an auto-numbering format.
Of course, for creating a link to an image, the user also still has the possibility to select an existing image file via the *Finding image file...* button.
2. If you chose *From field content*, the options in the *File name generation* box are read-only; with this input method file names cannot be generated automatically.
For *Automatic sequential generation*, setting the *File name generation* is mandatory, because file names of images to be read must be generated and saved automatically.
The file name generation uses auto-numbering to be able to generate a new file name for each new file. The file name is put together here in a similar way to autonumbered fields.

After setting these properties, your application is ready to import images through WIA.

Creating and importing images through WIA, from within Adlib

1. Connect your WIA compatible device to the computer, if that hasn't been done yet.
2. In your Adlib application, open the database in which the adjusted Image field appears, for instance *Visual documentation*.
3. Open a new or existing record to which you want to link a

reproduction image yet to be created, and place the cursor in the Image field, for instance *Identifier (URL)* in older applications or *Reproduction reference* in newer applications.

4. Click the *From scanner or camera...* button in the toolbar.
5. If you had set the *File name assignment type* for this field to *From field content*, then a standard window appears: *Save image as...* Search the desired folder to save the new image in, type a file name, choose a file type, and click *Save*.
If you had set the *File name assignment type* to *Automatic sequential generation*, then the *Save image as...* window is skipped because the file name will be generated automatically.
6. A WIA device dependent dialog appears with which you can make the new image, have it saved automatically, and directly include the appropriate link in the *Identifier (URL)* or *Reproduction reference* field. The created image is also immediately visible in the *Image viewer*.
See the documentation of the relevant device for an explanation of the options that the interface offers when making the image.

* Whether a device supports WIA actually depends on its driver software. Driver software (which can be updated), is either provided by Microsoft and is built into the Windows operating system by default, and/or provided by the manufacturer of the device. If the documentation of your imaging device doesn't mention WIA support, then please contact the relevant manufacturer (e.g. through their web site) and inquire about a driver supporting WIA.

8.9 Using a web browser control

From 6.6.0 you can place a special web browser box on a screen in your application, to display record data as a web page. The box is only meant for display, and possibly printing, but not for editing. You do need an XSLT stylesheet for this: under the hood of Adlib, all data is processed as XML, and with an XSLT stylesheet XML can be transformed to HTML, for example, while HTML can be displayed as a web page in a web browser and in the web browser box in Adlib. Such a web browser box can be useful when you want to present the user of Adlib with a nice presentation of the record in a single box. Or maybe you have a website which displays records in detail, and you would like to have the same presentation available in Adlib as well, so that during record entry a registrar can already see how the record will look on the website. Every time the screen is redrawn during entry or editing, for instance when you switch tabs, the web

page display will be updated. You can implement a web browser box as follows:

1. Create an XSLT stylesheet to transform Adlib XML of records from a certain database into HTML. Assume this is grouped XML as generated by the Adlib API too (also see the paragraph about the XML format below). Download the Museum example stylesheet (*MuseumRecordInWebBrowserControl.xslt*) from the Adlib website, or use the Adlib Office Connect stylesheets to start with. The example file contains an assumed, fixed relative path to the images folder: you should change this path to the (path to the) folder which actually contains your images. You are of course free to further adjust these stylesheets to your requirements. Place the stylesheet in a suitable location in your Adlib Software folder, maybe the folder with the name of the application, or in a new *\stylesheets* subfolder
2. Suppose you would like to be able to view an object record as a web page, then search your object catalogue for a screen with enough empty space to hold the web page presentation, or create a new screen. In this example we use the *Documentation (free)* screen (*docfarch.fmt*). Open the selected screen in the *Screen editor*.
3. Click *Insert > Web browser control* in the menu bar. A box with the icon of a globe will be placed on the screen. Drag it to the desired spot and make the box as big as you like by dragging its edges. When the HTML page is shown on this screen in your Adlib application, the box will have this exact size and cannot be adjusted there.
4. Right-click the new box and choose *Properties* in the pop-up menu which opens. There's only one option available. Click the ... button to search for the desired XSLT stylesheet on your system. The path to the file must be relative to the application folder, and the *.xslt* extension must be present behind the file name.
5. Save the changes in the screen and view the result in Adlib by opening a record and switching to the adjusted tab.

You can copy the web page display if you want, and paste it in a Word document. Right-click the display and choose *Select all*. Press **Ctrl+C** to copy everything. Switch to your Word document and paste the text (**Ctrl+V**).

If you like, you can even print the web page display directly. Right-click the display and choose *Print...* in the pop-up menu. With the standard Windows print dialog you can then actually start the printing.

It is possible that on the printout a page numbering, date and other header and footer information can be seen, and that background

colours won't print. This is caused by page settings in Internet Explorer, because the web browser box uses IE functionality for printing. In Internet Explorer 9 you can change these settings via the menu (press **Alt**) *File > Page setup* or via the *Tools* button (**Alt+X**) *Print > Page setup*. For example, mark the *Print background colours and images* option to be able to print background colours. The headers and footers can be changed here as well.

The XML format and the generated HTML

After setup, it is easy to find out how the record XML is structured exactly. You don't need to have an advanced stylesheet already or even one that works, to be able to view the XML. Right-click the web browser box in your active Adlib application and choose *View > Record XML* in the pop-up menu. The entire contents of the record will be displayed as XML in Windows NotePad. To obtain a better presentation of the XML, you can save the file with the *.xml* extension, and double-click it in Windows Explorer. The file will open in a program able to display XML properly, like Internet Explorer. Now, the XML structure is clear and you'll be able to customize your stylesheet efficiently.

Via the same pop-up menu you can display the generated HTML as well. Your stylesheet converts record XML into HTML and the result can be shown with *View > Page source*. Correct HTML code will be presented only if your stylesheet works properly.

System variables which can be used in the stylesheet

When the detail screen with the web browser control is formatted for display, three parameters are passed to the XSLT stylesheet: the currently selected data language, the current user interface language and the background color of the Adlib screen. These variables must be used as follows:

- `data_language` – the currently selected data language as an IETF language tag (as also used by Adlib in other multilingual functionality). From Adlib 7.0.0.106, the language tag is a code put together from an abbreviation for a language and a region identifier: for more information about this, see the "Using Language Identifiers (RFC 3066)" document which you can find on the internet. The code for British English, for example, is `en-GB`. In a stylesheet for a multilingual Adlib SQL or Adlib Oracle database you can use this parameter to select the data language in which you want to display field contents. See the *MultilingualRecordInWebBrowserControl.xslt* example stylesheet for the code behind a possible application of this functionality, for a multilingual *page_title* field.

- `ui_language` – the current user interface language as referenced in Adlib. For example, English is 0, while Dutch is 1. You can use this to present fixed texts in the display in the current interface language. See the *MuseumRecordInWebBrowserControl.xslt* example stylesheet for a way to use this parameter.
- `background_color` – the background colour of the screen as a hexadecimal HTML colour code (`#rrggbb`). You can use this parameter to provide the HTML page with the same background colour as the Adlib screen, if desired.

To use the parameters in a stylesheet, declare them as a regular XSLT parameter without a default value (because it will be overwritten anyway) somewhere in the file, for example:

```
<xsl:param name="background_color"></xsl:param>
<xsl:param name="data_language"></xsl:param>
<xsl:param name="ui_language"></xsl:param>
```

The background colour can be used like this, for example:

```
<style type="text/css">
  body { background: <xsl:value-of select="$background_color"/>
}; }
</style>
```

Error handling

Any errors in the stylesheet will be shown inside of the web browser control with line and column numbers, and an explanation for the failure.

Testing a stylesheet that you are still working on, is very easy. Once you've set it up like explained above, and you are currently showing a record in detailed display with the web browser box, you only need to switch tabs to reload the stylesheet. So after every change in your stylesheet, save it, switch tabs in Adlib, and you can see the effect of your latest adjustment immediately

Note

Every XSLT stylesheet for the current application will contain fixed HTML code. That means you can also add URLs to web pages. Use `target="_blank"` in the reference to have the link opened in your default web browser. If you leave `target="_blank"` out, the web page will open in the Adlib web browser box. Example:

```
<a target="_blank" href="http://www.adlibsoft.com">Adlib</a>
```

8.10 Using an HTML field

From 6.6.0 you can place an HTML field on a screen in your application. An HTML field is a database field meant for long, laid-out text. Layout can be applied to the text during editing of the record. You can print the contents of such a field to a Word template or with the aid of an XSLT stylesheet, whilst keeping the layout intact. Although you will see just the laid-out text while you are editing an HTML field, the field contents will actually be stored as HTML code in the background. You can use the field as an alternative for the older and rather narrowly applicable Rich text field type. From 6.6.0 you can implement HTML fields in your application.

For example: a good spot to implement such a field is the field for label text on the *Accompanying texts* tab of a museum object record (4.2 model application), since it would be nice if you could apply layout to a label text during data entry, and maintain that layout in printouts.

You can execute the following procedure in your own application, even if the field to be converted already contains data: your data won't get lost:

1. Create a backup of your database(s) and applications before you make any changes, just to be safe.
2. Open the *Collect* database in the *Application browser* in Designer, and select the *label.text* field (tag AB).
If this field is not present in your application, you can of course pick another field of the *Text* data type to convert, or create a new field.
3. Open the *Data type* drop-down list on the *Field properties* tab (which has *Text* currently selected) and choose the *HTML* option.
4. Save the change: right-click the *Collect* database and select *Save* in the pop-up menu.
5. In the *Screen editor*, open the *labels.fmt* screen (*Accompanying texts*), or another screen on which you want to place the new HTML field.
6. Drag the lower border of the *Labels* box a few centimetres downwards so that some space is created for the new field. In the menu choose *Insert > HTML field*.
7. Drag the HTML field into the box to the desired location and make it as big as you like. This type of field will not automatically resize as the user types more text in it, but a scroll bar will appear instead. So if there is enough room on the screen, like in this

example, then make the HTML field tall enough to allow long text to be visible in its entirety, in most cases. If desired, you can insert a label in front of the HTML field, into which you copy label texts from the already existing *Text* field label.

8. Now select the old *AB* field and choose *Edit > Delete* in the menu. Label and field will be removed and the fields underneath it will move up a line.
9. Right-click the right or left border of the HTML field (where the cursor changes into a double arrow) and choose *Properties* in the pop-up menu. (See that you do not open the properties of the *Box* by accident.) Now enter the *Tag* of the HTML field, *AB* in our example, and set the other properties as desired. In this example, the field should be set to *Repeated* because the old field was repeatable too.
10. Save the changes in the screen and you are done.

Restart Adlib to be able to admire the result. Any existing text has no layout yet and hasn't been stored as HTML code either. The record has to be put in edit mode to apply layout via the buttons in a floating toolbar. Note that existing text in this converted field will only be saved as HTML when the user has placed the cursor in this field once and saved the record again.

Printing, export and wwwopac output

The laid out text in HTML fields can only be printed correctly to Word templates or XSLT stylesheets. In principle, you must create these templates or stylesheets yourself, but templates do not require any special instructions – you can refer to the field tag normally – and an example stylesheet is available for download [here](#); you may still have to adjust the stylesheet to your own situation. A stylesheet must be set up as an output format in the proper data source, before the user can print to it from within the *Print wizard*.

If you still print HTML fields via an *adapl* (also via the interactive *Print wizard* method), then the HTML field contents will be printed: the layout won't be visible but the HTML codes will be.

When you export records with HTML fields, the HTML field contents will be extracted as HTML code: the code begins and ends with `<div>` and respectively `</div>`, and has no `<body>` or `<head>` tags. When exporting to XML, the HTML field contents will be produced as HTML within the XML field tags; the same applies to the XML search result of *wwwopac*. An Adlib Internet Server uses stylesheets to convert the XML output of *wwwopac* to HTML.

For both printing and web display, XSLT stylesheets need to be coded for HTML fields in such a way that the HTML field contents will be

copied from the XML literally, like for the `label.text` field in our example:

```
<xsl:template match="label.text">
  <xsl:copy-of select="."/>
</xsl:template>
```

Notes

- Existing, filled-in RTF fields cannot be converted to HTML fields.
- After the conversion, hard and soft returns in existing text fields have been converted to `
` tags.

8.11 Showing the upwards hierarchy of a linked term

When you register the location of an object in your collection, it is sometimes handy to have an overview of the upwards hierarchy of that location displayed in the record as well. From 6.6.0, it is possible to implement this functionality for all fields which link to an internally linked field. You'll have to make some changes in your Adlib application to set this functionality up:

1. Open the Application browser.
2. In the database which holds the relevant linked field, for instance the *Collect* database, create a new field of the data type *Temporary* or *Text*, which is sufficiently long to contain all the terms of your most extensive term hierarchy. Use a unique field tag and choose a meaningful field name. If you make it a *Text* field, then also mark the *Do not show in lists* option for the field.
3. In the *Context* property on the *Relations* tab in the field properties of the linked field in the catalogue (which links to a field which in turn is internally linked to broader and narrower term fields), for example the *location.default* field, enter the tag or field name of the new field. Save the changes in this database.
4. Place the new field somewhere on a screen, and make it read-only since it is only for display. In our example, the *location.context* field would fit well on the *Location | Future movements* screen (*location.fmt*) in the 4.2 application. You could name the label for the new field *Location hierarchy*, for example. Save your changes.

Restart your Adlib application and observe the results. Adlwin will automatically fill the new field with the hierarchical context of the

linked term, yet only its broader terms and the term itself, in the following syntax: *top_term/bt/bt/.../bt/term*. So the hierarchy is presented as a single string, its broader terms separated by slashes.

If an entered location appears more than once in the database, which may easily be the case for general location names like "shelf 1" or "rack 3", you must select the right term when you leave the field. The *More than one preferred term found, select from list* window opens automatically, listing the identical terms. Hover the mouse pointer over a term, and a tooltip should appear, showing the upwards hierarchy of the term. Do this over every term until you find the right one. Then select it and click *OK*.

In the linked field, you can also press **Shift+F4** to select the desired term in the *Find data for the field...* window.

8.12 Character set conversion of your data and/or application

Each file that contains text in some form, is encoded in a specific character set, although the average computer user is normally not aware of this. The character set determines which characters can be used in the text, and implicitly determines the compatibility of the file with all kinds of software.

In Adlib it's important to distinguish the encoding of your data (the *.cbf* file and its indexes) and all other Adlib files, like the application structure (*.pbk*) or database definitions (*.inf*).

Change the encoding of Adlib structure and settings files

The *Encoding* property of Adlib structure and settings files, displays the type of character set used to encode texts that you provide for properties of the object/file. Different files may have a different encoding, but to prevent confusion about what characters can be used where, it is of course best if all Adlib structure and settings files use the same character set. You can change the setting for an individual file manually, if desired.

You can change the encoding of database structures (*.inf* files), application structures (*.pbk* files) and screens (*.fmt* files) also simultaneously, with the *Application character set conversion* tool in Adlib Designer. (The currently used character set of a *.pbk*, *.inf* or *.fmt* file is displayed on the *Properties* tab under *Encoding* when you select such a file in the *Application browser*.)

Start the *Application character set conversion* tool by choosing *Tools >*

Application character set conversion in the main *Adlib Designer* window. With this tool you can convert your Adlib application (only *.pbk*, *.inf*, and *.fmt* files) to any of three character sets. But probably you will only use it to convert a DOS (OEM) application to either an ISO-Latin (ANSI) or Unicode one, or to convert an ISO-Latin application to Unicode. Note again that this tool only converts the application and database structures, not your databases and data.

You only need to consider using this tool if your language has accented or other special characters and you need to use those in for instance the labels of fields, or in title bars. Once converted to another (expanded) character set, you'll be able to use characters from it in your application setup.

Before converting, make sure you select the proper work folder. Since you probably want the change to affect all your Adlib modules, you must select your main (copy of an) Adlib folder, and mark the *Convert subfolders* option in this tool window.

Note that a Unicode application may cause problems if you want to edit it in one of the old DOS tools ADSETUP or DBSETUP, because Unicode characters cannot be displayed in there.

Also note that the ANSI version of *adlwin.exe*, can run Unicode applications that work on ANSI or DOS databases.

Change the encoding of your data

Your data is stored in databases (*.cbf* files) and the derivative indexes. If you need to be able to enter different characters than the current encoding allows, you may consider a conversion of your databases. The current encoding of your data can be found in the *Locale* property of each database.

The following guidelines can be given for changing this property for existing databases:

- The encoding of the data can be different from the encoding of Adlib structure and settings files.
- The value of the *Locale* option must be the same for all databases in a single Adlib application, otherwise an error will occur.
- Simply change *DOS* to *ISO-Latin* if you want to be able to enter characters like the euro-sign €, but are not interested in characters from languages like Hebrew, Chinese, etc. All characters in ISO-Latin (more specifically WinLatin1-cp1252) that are extra to DOS, are the following:

80	€		82	,	83	f	84	,,	85	...	86	†	87	‡	88	^	89	%	90	Š	91	Š	92	Š	93	Š	94	Š	95	Š	96	Š	97	Š	98	Š	99	Š																																	
	91	€	92	,	93	f	94	,,	95	...	96	†	97	‡	98	^	99	%	9A	Š	9B	Š	9C	Š	9D	Š	9E	Š	9F	Š	9G	Š	9H	Š	9I	Š	9J	Š	9K	Š	9L	Š	9M	Š	9N	Š	9O	Š	9P	Š	9Q	Š	9R	Š	9S	Š	9T	Š	9U	Š	9V	Š	9W	Š	9X	Š	9Y	Š	9Z	Š	
A0	A1	i	A3	φ	A4	φ	A5	φ	A6	φ	A7	φ	A8	φ	A9	φ	AA	φ	AB	φ	AC	φ	AD	φ	AE	φ	AF	φ	AG	φ	AH	φ	AI	φ	AJ	φ	AK	φ	AL	φ	AM	φ	AN	φ	AO	φ	AP	φ	AQ	φ	AR	φ	AS	φ	AT	φ	AU	φ	AV	φ	AW	φ	AX	φ	AY	φ	AZ	φ			
B0	°	B1	±	B2	2	B3	3	B4	-	B5	μ	B6	¶	B7	.	B8	,	B9	1	BA	2	BB	3	BC	4	BD	5	BE	6	BF	7	CG	8	CH	9	CI	0	CJ	1	CK	2	CL	3	CM	4	CN	5	CO	6	CP	7	CQ	8	CR	9	CS	0	CT	1	CU	2	CV	3	CW	4	CX	5	CY	6	CZ	7
C0	À	C1	Á	C2	Â	C3	Ã	C4	Ä	C5	Å	C6	Æ	C7	Ç	C8	È	C9	É	CA	Ê	CB	Ë	CC	Ì	CD	Í	CE	Î	CF	Ï	CG	Ð	CH	Ñ	CI	Ò	CJ	Ó	CK	Ô	CL	Õ	CM	Ö	CN	×	CO	Ø	CP	Ù	CQ	Ú	CR	Û	CS	Ü	CT	Ý	CU	Þ	CV	ß								
D0	Ð	D1	Ñ	D2	Ò	D3	Ó	D4	Ô	D5	Õ	D6	Ö	D7	×	D8	Ø	D9	Ù	DA	Ú	DB	Û	DC	Ü	DD	Ý	DE	Þ	DF	ß	EG	à	EH	á	EI	â	EJ	ã	EK	ä	EL	å	EM	æ	EN	ç	EO	è	EP	é	EQ	ê	ER	ë	ES	ì	ET	í	EU	î	EV	ï	EW	ð	EX	ñ	EY	ò	EZ	ó
E0	à	E1	á	E2	â	E3	ã	E4	ä	E5	å	E6	æ	E7	ç	E8	è	E9	é	EA	ê	EB	ë	EC	ì	ED	í	EE	î	EF	ï	EG	ð	EH	ñ	EI	ò	EJ	ó	EK	ô	EL	õ	EM	ö	EN	÷	EO	ø	EP	ù	EQ	ú	ER	û	ES	ü	ET	ý	EU	þ	EV	ÿ								
F0	ä	F1	ñ	F2	õ	F3	ó	F4	ô	F5	õ	F6	ö	F7	÷	F8	ø	F9	ù	FA	ú	FB	û	FC	ü	FD	ý	FE	þ	FF	ÿ	FG		FH		FI		FJ		FK		FL		FM		FN		FO		FP		FQ		FR		FS		FT		FU		FV		FW		FX		FY		FZ	

You need no special conversion of your data because the DOS character set is a subset (the first half) of the entire ISO-Latin set (the image above only displays the second half).

- You need Unicode (UTF-8 representation) encoding, if you want to be able to enter characters that do not occur in either DOS or ISO-Latin, and/or use the sort and search order of your language region (see the description of *Locale*). This is not just a matter of changing this setting, unfortunately. It involves an elaborate conversion. Ask Adlib for more information.
Note that if your databases are encoded in (UTF-8) Unicode, you'll need adlwin.exe version 6.0 or higher to run your Adlib application.

Conversion of data during import - If you are about to import an existing DOS database into your ANSI (ISO-Latin) or UTF-9 Unicode encoded data, you must convert the imported records to the proper encoding explicitly (from Adlib version 6.2.0). In the field mapping of an import job you must then use a special tag pair to execute this character set conversion of your data during import (either from adlwin applications or designer). As source tag, enter: `DOS`, and as destination tag, enter: `<CHARSET>`. (Type both terms literally as printed here.)

If you are about to import an existing ANSI encoded database into a UTF-8 encoded database, then enter as the source tag: `ANSI`, and as the destination tag, again: `<CHARSET>`. (This particular tag pair is available from Adlib version 6.2.0.) Also, remove the *wordlist.idx* file before importing: it will be rebuilt automatically during import, in the new encoding.

The third possible conversion is from OCLC to the encoding of the target database (ANSI or UTF-8 Unicode). The relevant tag pair is: `OCLC <CHARSET>`.

Conversion of data after export - Data will always be exported from Adlib to UTF-8 encoded Unicode files, regardless of the encoding of your databases. If you want these exchange files to be encoded differently, you'll have to do that separately in Windows NotePad or WordPad. Just open the exchange file in the appropriate text editor and save it in the desired encoding.

See also

Inserting special characters in text

8.13 Inserting special characters in text

In the texts you write for use in Adlib, you can use special characters (possibly from other languages) which come in the character set, set up for your particular Windows installation.

The easiest way to insert a special character in a text is as follows:

1. Place the cursor in the text where you want to place the special character.
2. Open the Windows *Start* menu.
3. Choose *Programs > Accessories > System Tools > Character Map*.
4. Make sure the Windows *Character set* for your language area has been set correctly. For western Europe for instance, this will have to be *Windows: Western*, if you want to insert the characters into an ANSI version of Adlib. If you work under Windows 2000 or XP and you have the Unicode version of Adlib you may also choose *Unicode* as the character set; this will allow you to choose from far more special characters, like those from languages as Hebrew, Japanese or Greek for example.
5. In this window also choose a *Font*, because different fonts not only display characters differently, but some fonts have more characters than others.
6. Now click the desired character, *Select* and *Copy* it, return to your document and paste the copied character.

The fastest way though, to insert special characters is through a key combination of the left **Alt**-key and a four digit number from the numeric keypad. Not all special characters have such shortcuts, but for example € does: **Alt-0128**. When you are in the *Character Map* utility as described above, you'll find the shortcut (if present) for a selected character in the status bar of this window, in the right corner.

8.14 Working with non-standard dates

In Microsoft Windows, calendar controls cannot generate dates earlier than September 1752.

In Adlib these controls are used for searching and data entry of dates:

- For searching, you cannot use a date index to search for earlier dates, because date indexes are accessed by means of calendar controls.
- For entering an earlier date in a record, the *Data type* of the entry field (on the screen) cannot be *Date*, because this also uses a calendar display.

Another problem might be that sometimes you would like to enter incomplete dates, because only the year is known, or periods, or negative dates.

Solution

The date field in the database can in principle hold any date, but the problem is accessing that field from the Adlib interface.

For data entry, that means defining the data type of the entry field on screen as *Text*, so that no calendar will be used.

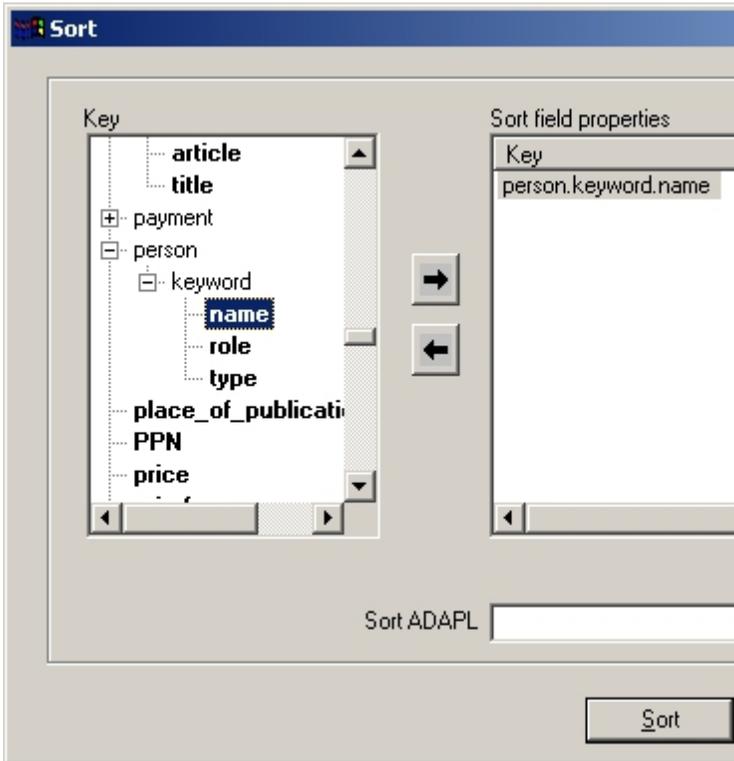
For searching, it means creating a text (term) index instead of a date index. In a term index you can search without being hindered by the limited calendar control. But the only useful way of searching is when the data type of the date field in the data dictionary has been set to *ISO date (yyyy-mm-dd)*. This has to do with the alphabetical (!) sorting of a term index: alphabetical sorting on European or American dates will sort on day or month, making it impossible to find a year.

An added advantage to using this setup is that you'll be able to enter incomplete or negative dates.

If your database is already filled however (especially the concerning date field of course), you can't just change the data type of the data dictionary field, because it would result in errors (Adlib doesn't automatically convert existing content). So you'll have to do a data conversion for the concerning field, for instance by using a stand-alone *adapl* that will have to be written. You must change the data type of the data dictionary field directly before such a conversion. And then you'll have to change the index definition for that field, and rebuild the index.

8.15 Naming fields for hierarchical display

The field lists in the *Sort* window, the *Expert search system*, the *Replace* window and in the *Export* wizard can be displayed in the form of a hierarchic tree structure. This has nothing to do with broader and narrower terms, but is just intended to make the list of fields more organized, by grouping field names that "belong together". See the figure below for an example.



Only bold printed names represent fields that the user can select, although the entire field name may include higher nodes that are not printed in bold type. This is to make a distinction between higher nodes that themselves are selectable fields. In this example you can't select *person* or *keyword* because *person* and *person.keyword* aren't fields; if they were, they would be printed in bold type too.

The hierarchy in a field list is determined entirely by how data

dictionary fields are named. In a field name a dot (.) is used for this purpose: simply separate words in a field name by a dot to let Adlib display this field name in a hierarchy, as can be seen in the example of the field name *person.keyword.name* above. If you want to group other fields under *person > keyword*, their names must also begin with *person.keyword*. For every language in which you want to present field names hierarchically, you have to name fields using dots.

Note that in field names you may also encounter underscores (_) to separate words in the name. This has no special meaning, it's just to make the field name better readable since spaces aren't allowed.

In general, you can change field names in your existing databases without getting into trouble, because although many properties in Designer can hold tags as well as field names, it is always the tag that is stored. The only location where field names may be stored instead of tags, is in Word templates. So if you change field names, check your Word templates for old field names that need to be changed too.

8.16 Regular expressions

A regular expression is sort of a template that prescribes how the contents of a field should be formatted. You can use it to indicate what characters Adlib should accept in the input and what characters should appear at what position.

In Adlib, regular expressions can be set in the properties of entry fields on screens, to limit user input for those fields.

The following format codes are available:

Character(s)	Meaning
.	Any character may be entered by the user at the position of this character in the regular expression. For example: means that the user may enter four random characters.
*	The preceding character or expression may occur zero or more times. For example: [0-9]* indicates that any number from 0 through 9 may occur zero or more times at the position of * in the regular expression.

+	The preceding character or expression must occur at least once but may also occur more times. For example: [b-f]+ indicates that any letter from b through f may occur one or more times at the position of + in the regular expression.
?	The preceding character may occur 0 or 1 times at the position of ? in the regular expression. For example: !? means that ! may occur here once, or not at all.
[characters]	Insert all the characters that are allowable for input by the user at this position in the regular expression, between the brackets, without spaces in between the characters, for example: [aHgm].
[x-y]	The allowed input of one character falls within the range x - y, for instance [a-z] or [0-9]. More than one range can also be provided, by placing them directly behind each other between the brackets, for example: [0-9a-z].
[^characters]	Insert all the characters or a range between the brackets and after the '^' symbol that may not occur at this position. For example: [^0-9] which means that the numbers 0 through 9 may not occur at this position.
^	Only when this character is inserted at the very beginning of a regular expression, it means that input should start with the expression or characters following this character.
\$	This character indicates the end of the allowable input. No more characters may be entered by the user.
\character	The character that may occur at this position is a special character reserved for use in a regular

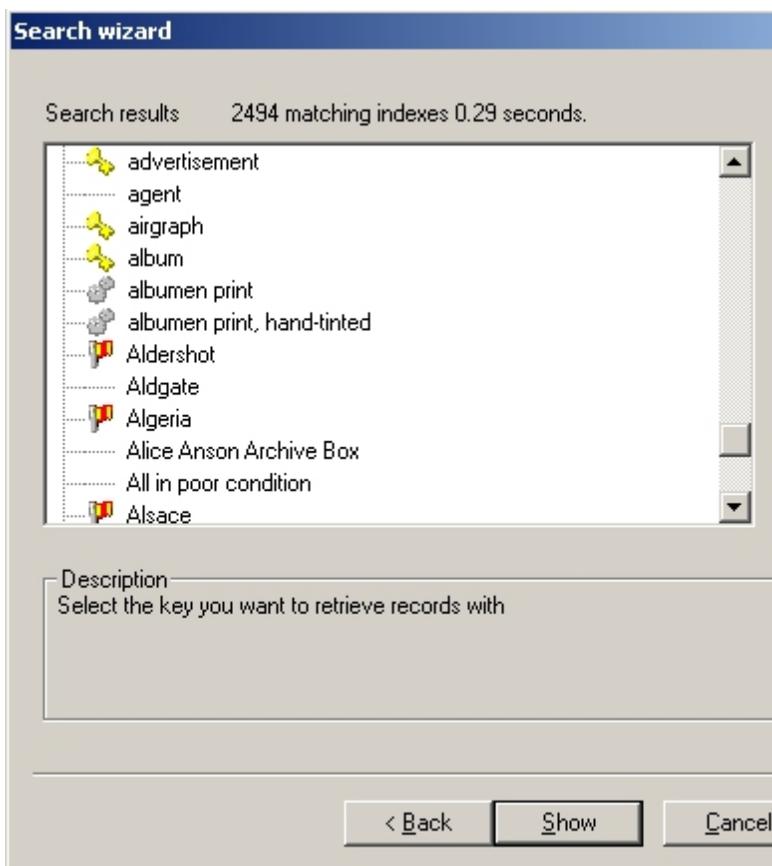
	<p>expression. An example is the full stop, which is used in regular expressions as a wildcard. If you want the user to be able to enter a full stop as a character, place it behind a backslash: \.</p> <p>Another example is \? * \^ which means that the user may either enter a ?, a * or a ^.</p>
expression 1 expression 2	<p>Allowed input either matches expression1 or expression2. The symbol for OR is a vertical line. For example: [0-9] x means that the user may enter either a number or an x.</p>

Below you'll find some more examples of how to use regular expressions:

Character(s)	Allowed input
h..se	an h followed by any two characters, followed by se. This would match horse and house, for example.
The.*house	the text The followed by any one or more characters, followed by house. This would match: The house, The new house, The boathouse, but not Thehouse.
[Ee]xample	the word example, spelt with either a capital or a lower case e.
l[aeiou]nger	a word with a vowel at the second position. For example: longer, linger, etc.
[1-9][0-9][0-9][0-9] [A-Z][A-Z]	the regular expression for Dutch postcodes. Allowed values are: 1234HK or 9000ZA, but not 0434AB or 3456hj.
book serial	the word book or the word serial.

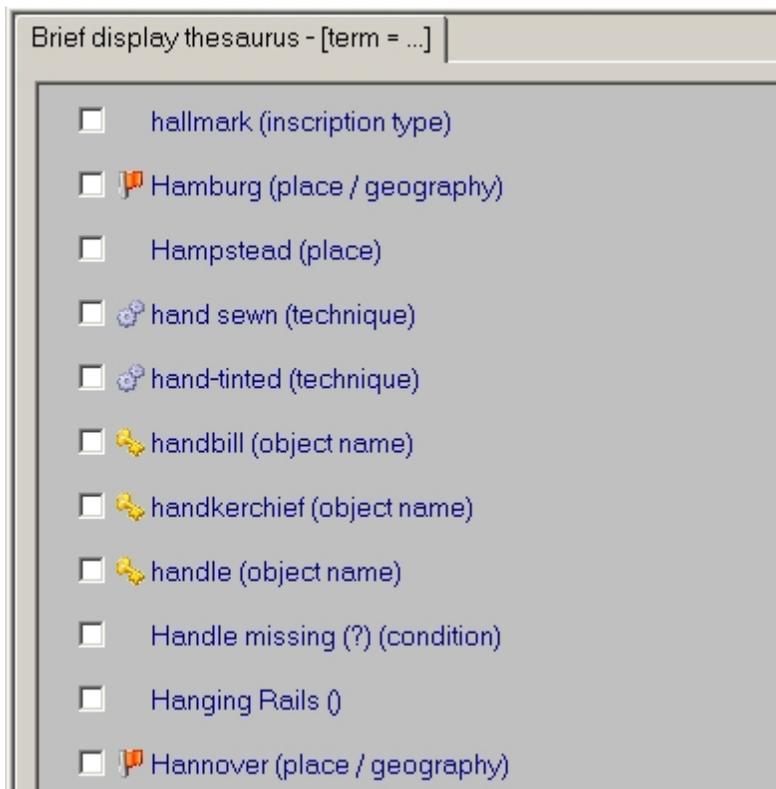
8.17 Domain-specific icons

It is possible to assign icons to domains. Such icons are displayed in Adlib in lists of retrieved keys (in the *Search wizard* and the *Linked record search screen*), and in the search result on a brief display screen for an authority file, in front of the term from that domain. You could use this functionality to emphasize terms from certain domains, for easier recognition of those terms. It is probably not a good idea though, to link icons to all domains, because that would more likely cause visual chaos.

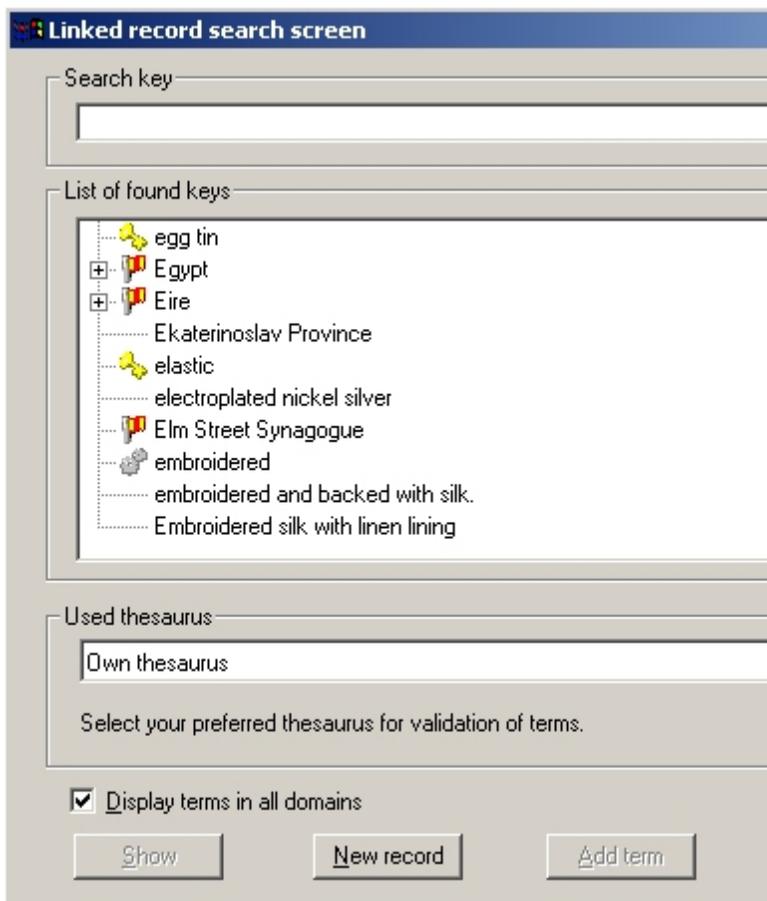


You need an icon (a special image file with the extension *.ico* *) for each domain that you wish to indicate this way. Store each *.ico* file with the name of the domain it belongs to (not case-sensitive), in the application directory.

The names of domains (the so-called neutral values) can be found through the *Application browser*. Open the *\data* folder and select the domain field in the desired authority file, e.g. *term.type (do)* in the *thesau*, or *name.type (do)* in the *people* database, and switch to the *Enumeration values* tab for a list of all neutral values.



To obtain the screenshots on this Help page, three icons were placed in the *\standard* folder of a Museum Plus application. The icons were named: *OBJECT.ico* (the yellow key, which represents the domain *object name* here), *TECHN.ico* (the grey gears, which represent the domain *technique* here), and *geokeyw.ico* (the orange flag, which represents the domain *geographical keyword* here).



* Icons can be created in professional graphics software or specialized icon designer programs. If you want to have a try at it yourself, you could download some shareware for this purpose. But for professional icons you'll probably need a graphical designer to do it for you. Alternatively, you can search the internet for ready-made icons.

8.18 HTML start page for data source selection



From Adlib 6.6.0, this is deprecated functionality. From Adlib 7.0.0, the user can no longer switch between modes and from Adlib 7.1.0 the dashboard functionality is not supported anymore, meaning that it can't be set in Designer anymore and that earlier set up dashboards will be ignored by Adlib. So the following text does not apply to Adlib 7.1.0 and higher.

After starting Adlib, normally the *Search wizard* will be opened, in which you choose the data source (database or dataset) to search in or edit records in. This interface, *Step 1 of 4 of the Search wizard*, with an empty background, is called the classic mode. In the *Options* menu (of Adlib 6.6.0 and older software) in a running Adlib application you can switch this mode on or off. (Adlib needs to be restarted before a mode switch comes into effect.)

The new mode (classic mode is switched off) replaces *Step 1* of the *Search wizard* with a dynamically generated HTML page called the dashboard. Data sources and procedures (like print tasks) are presented as hyperlinks which need to be clicked to open the relevant database or start the procedure. *Step 2* of the *Search wizard* will not be presented next automatically anymore. Instead, the user must explicitly choose the search method to continue his or her search with, via the toolbar or menu: this may be the *Search wizard*, the *Expert search language*, a QBF (search form) or the *Pointer files* window.

In Adlib applications older than version 4.2, the dashboard looks very simple by default because it is put together by the software automatically. In model applications 4.2 and higher, stylesheets have been implemented to provide a nice design for the dashboard. (Adlib produces the dashboard by applying an XSLT stylesheet to the data sources list generated in XML format.)

However, you can implement the dashboards from model applications 4.2 in older applications very well. You do need some knowledge of the programming language XSLT, XML and possibly CSS to be able to do this well. Contact our helpdesk to obtain the relevant dashboard stylesheets. The package consist of several stylesheets (.xsl files), *dashboardLanguage.xml* (fixed texts in different languages), *Dashboard.css* (layout styles) and some images. Implement the stylesheets in an application, as follows:

1. Unpack the files, if they are still contained in one zip file.
2. Collect the files into one new subfolder called `\dashboard`, underneath your Adlib main folder.

3. Open the .pbk file of your application or module in the Application browser.
4. Open the *Advanced* screen tab.
5. In the *Adlib menu style* entry field, enter the (relative) path to the desired stylesheet. Each stylesheet has been created specifically for one application or module, as you can see from their names. Click the ... button to search for the file on your system, and change the absolute path into a relative path (relative with respect to the pbk file).
6. Repeat steps 3 up to and including 5 for all applications and modules (if desired). Save all changes.

The stylesheets have now been linked, but they are not likely to function properly yet because the list of data sources and print tasks in your application probably differs from the list in the Adlib model application 4.2. Therefore you have to adjust the stylesheets to your applications. You may as well change the entire design if you want to, and include the logo of your institution, for example. It is beyond the scope of this text to explain all possibilities of XSLT, but it is relatively easy to adjust the data source lists, as follows.

Open an .xsl file which you want to adjust, in Windows Notepad for example. The data source lists are present in `<div id="menu">` elements. In *DashboardArchiveMuseum.xsl* for instance, there are five lists. For the supporting databases, the list is the following:

```
<div id="menu1">
<ul>
<h2>
<xsl:call-template name="translate">
<xsl:with-param name="label">SupportDatabases</xsl:with-param>
<xsl:with-param name="language" select="$language" />
</xsl:call-template>
</h2>
<xsl:apply-templates select="//*/item[@id='12']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='19']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='20']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='21']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='22']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='23']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='24']" mode="table"/>
<xsl:apply-templates select="//*/item[@id='25']" mode="table"/>
</ul>
</div>
```

Each @id= points to the serial number of a data source in the XML list.

This is the same order (from top to bottom) as the one you can observe in *Step 1* of the *Search wizard*. The top most data source (e.g. *Total collection*) is number 1, the next is number 2 etc. In the example above, the number 12 is at the top of this list. In our case, that is the *Visual documentation* data source. You can change this and just replace "12" by another serial number, to point to a different data source. You may also add lines to this list or remove lines. The fixed texts in the dashboard can be changed in *dashboardLanguage.xml*. The reference to a specific text in the example above can be found in:

```
<xsl:call-template name="translate">
  <xsl:with-param name="label">SupportDatabases</xsl:with-param>
  <xsl:with-param name="language" select="$language" />
</xsl:call-template>
```

8.19 Approaching external sources as friendly databases

Five so-called gateways (HTTP handlers) to external sources (third-party databases), are made freely available by Adlib. In your applications you can point to those external sources in the shape of friendly databases. This allows you to derive records from these sources, so you don't need to enter that data yourself anymore. The external sources which are available at the release of Adlib 6.5.0, and the accompanying URL to the gateway for each of them, are the following:

- **Koninklijke Bibliotheek (Dutch Royal Library):**
<http://gateway.adlibsoft.com/dutch-royal-library/gate.aspx?search=>
- **British Library:**
<http://gateway.adlibsoft.com/british-library/gate.aspx?search=>
- **Library of Congress:**
<http://gateway.adlibsoft.com/library-of-congress/gate.aspx?search=>
- **German National Bibliography:**
<http://gateway.adlibsoft.com/deutsche-nationalbibliothek/gate.aspx?search=>
- **Gemeinsamer Verbundkatalog (GVK - German)**

<http://gateway.adlibsoft.com/german-gvk/gate.aspx?search=>

Such external sources can be set up as friendly databases in Designer, either manually or semi-automatically. The manual way is as follows:

1. Create a friendly database in the data source in which you wish to be able to derive records from the external source, for instance the *Books* data source in a library catalogue.
2. In the *Folder* property of the friendly database, enter the URL followed by `?search=`
3. Per friendly database you have to set three applicable screens, one to be able to search the external source, one to display a found record and a link screen. Which fields (may) appear on those screens, depends on the relevant external source. We have already made these screens for the Koninklijke Bibliotheek, the British Library, the Library of Congress, the German National Bibliography and the GVK. Contact Axiell ALM Netherlands to receive these screens by e-mail. Per external source you'll get three different screens; for the Library of Congress for instance, these would be *qbf_loc*, *lnk_loc* and *zm_loc*. First, put all received screens in your own Adlib \screens subfolder. Then set the proper three screens in the *Screens* properties of the friendly database, for example:

Search screen: ../screens/qbf_loc

Search result screen: ../screens/lnk_loc

Zoom screen: ../screens/zm_loc

(The screen names for the Koninklijke Bibliotheek end with "kb", those for the British Library end with "bl", those for the German National Bibliography end with "dnb" and "ext", while those for the GVK end with "gvk" and "ext".)

4. The *Remove original record after retrieval* option must of course be left unmarked, since you can't delete original records from the external sources.
5. Save the application structure and restart your Adlib application. You are now ready to derive title descriptions from the external sources you've just set up.

However, Adlib Designer can make these settings automatically as well:

1. Right-click the data source in which you wish to be able to derive records from the external source, for instance the *Books* data source in a library catalogue.
2. In the pop-up menu that opens, you should find the *Choose friendly databases* option. If this option is missing, Designer was

not able to retrieve the necessary data for making the settings from the internet, and you have to apply the manual settings anyway. Choose said option.

3. The *Select services from gateway.adlibsoft.com* window opens. Simply mark the external sources that you like to set up, and then close the window. The required screens will be downloaded automatically and set up as well (if the download doesn't succeed, you can continue only using the manual settings).
4. Save the application structure and restart your Adlib application. You are now ready to derive title descriptions from the external sources you've set up.

8.20 Requirements for the Change locations procedure

The *Change locations* procedure in your Adlib Museum application allows you to simultaneously change the current location of one or more marked object records in the Brief display, when you are actually moving objects.

The *Change locations* procedure (in Adlib 7.1 and higher) uses hard coded tags for its field mapping. If you want to customize an application, it might be handy to have an overview of the relevant tags. To start with, Adlib distinguishes between 4.2 model applications and older model applications. This distinction is made on the basis of tag 2A (*current_location*) in the database *collect*. If the tag is present, the software assumes that it concerns application version 4.2 or higher. So, this tag in that database should not have a different purpose, in whatever application, if you want to be able to use the *Change locations* procedure.

For application version 4.2 and higher, the following applies:

Field in <i>Change object locations</i> window	Tags to be filled by the procedure in database <i>collect</i>
<i>Location</i>	2A (<i>Current location</i>)
<i>Date/Time</i>	2C/2G, the start date and time of the new current location (and SE/Sh, the end date and time of the now previous location, if present)
<i>Suitability</i>	2E

<i>Authoriser</i>	2F
<i>Notes</i>	2D
<i>Method</i>	mT (<i>Movement method</i>)
<i>Reference</i>	mR
<i>Contact</i>	mC
<i>Notes</i>	mN
	2R (current executor)

When the procedure automatically transfers the details of the current location to a new first field group occurrence of the location history (which of course happens before the procedure enters a new current location), the following tag mapping is used in 4.2.

Current location fields	Location history fields
2A (<i>Current Location</i>)	ST
2C/2G (<i>Date/Time</i>)	SS/SH
2E (<i>Suitability</i>)	S3
2F (<i>Authoriser</i>)	SP
2D (<i>Notes</i>)	LM
2R (<i>current executor</i>)	2V (<i>executor history</i>)

In the 4.2 model application, 2R and 2V have not been defined in the database structure yet and are not visible on the screen (they will be in a future model application), but they will still be stored in the record. In principle that is not problem. If you want, you can define 2R and 2V as text fields with a length of maximally 100 characters in the *collect* database (2R not repeatable, 2V repeatable) and place them each on their applicable location screen, right above the *Authoriser* field for example, to make this data visible.

For application version 3.4 and older applies:

Field in <i>Change object locations</i> window	Tags to be filled by the procedure in database <i>collect</i>
<i>Location</i>	ST
<i>Location type</i>	LT
<i>Date</i>	SS, the start date of the new current

	location (and SE (2), the end date of the now previous location, if present)
Notes	LM

Before the procedure enters a new current location, a new, empty first occurrence of the field group for the location details is created. This means that the data from the old first occurrence simply moves down to the second occurrence, which makes the old current location the new previous location. The relevant tags in 3.4 are the following:

Current location fields	Location history fields
ST (1) (<i>Location</i>)	ST (2)
LT (1) (<i>Location type</i>)	LT (2)
SS (1) (<i>Start date</i>)	SS (2)
LM (1) (<i>Notes</i>)	LM (2)
S3 (1) (<i>Suitability</i>)	S3 (2)

8.21 Testing your application for errors

Adlib Designer has a special tool with which you can check your application for certain common errors. Start this *Application tester* tool by choosing *Tools > Application tester* or by clicking the button for it in the main *Adlib Designer* window :



This was originally a tool used by the Adlib developers to test their own work on Adlib Designer, by checking the files for compatibility and consistence. This was to make sure that Designer can read files from all versions of Adlib properly, and save them to disk (unmodified) the same way the old ADSETUP or DBSETUP tools would. These specialized tests can be found on the *Advanced** tab, and most users can disregard them.

However, on the *Options and file types* tab you'll find settings to perform a consistency test on your own application. This test checks the file types and work folders that you select on this tab, for:

- errors in path names (meaning references to Adlib objects, like screens and linked files), that point to missing files or typing errors in the path names;

- errors in link definitions (meaning the syntax of path names: if file separator characters are being used properly, although in Designer you no longer see these characters in object properties because paths to datasets are now split up in different properties);
- multiple (duplicate) use of tags and link reference tags in *.inf* files (because these must be unique);
- undefined tags which are being used in linked field definitions, like undefined link reference tags. Every tag you use, should be defined in the data dictionary.
- spaces in field names.

So, first select a work folder and mark the desired options, and then click the *Start* button to start the test. Click the *Stop* button if testing takes too long, and you want to cancel the process.

Feel free to try this tool, but do realize that you can't use it to test every aspect of your application on validity because the test is limited. This means you can't rely on this tool to tell you if your application and databases are built correctly if it reports no errors, but you can find certain errors.

Any errors that you do find using the *Advanced* tab of this tool, may be of interest to the Adlib developers, to improve Designer; in other cases you may be able to use an error report to improve your own Adlib application.

Progress and error report

The lower half of the *Application tester* displays progress information. If the tested files contain errors, you will be notified and a report is generated in the main *Designer* window. You may add remarks by typing them anywhere you wish. It's also possible to save this report as an *.rtf* file, that you can open in most text editors. If the report is long, you can search it for any term with **Ctrl+F**.

Non-modality of the application tester

The application tester stays active (and running if applicable) when you close it, which means that any errors are still being written to the main *Designer* window, and that if you open the tool again you'll see the results of the current/last search.

* Test the writing of files through options on the Advanced tab

Mark the *Write test* option to have the *Application tester* create temporary copies of the files in the current work folder, save them to disk and check them for consistency; only *.fmt* files (screens) are

excluded from this test. So the test is non-destructive, it doesn't change your existing files. The temporary copies are automatically removed after the test.

In case a write test fails, the developers of Adlib Designer may need the temporary copy of that file to analyse the problem. Mark the *Keep failing write test files* option to save those copies. They have the same name as the original file but with the extension *.object_test*.

8.22 Error codes

Code	Cause
0	No error, action successful
1	Tag not found
2	Memory allocation error
3	Overflow error
4	String too long
5	Error in date or in date format
6	Wrong key type
7	Record not found
8	Internal error
9	Error in database handle
10	Index not open
11	Database not open
12	Database read error
13	Database creation error
14	Database search error
15	Database insert error
16	Database info file not open

17	Error in database info file
18	Error writing to database info file
19	Dongle error
20	List empty
21	No index
22	Error making new priref
23	WORDLIST error
24	BTREE error
25	BTREE initialization error
26	Free text search error
27	Key length error
28	Error deleting data file
29	Key truncated error
30	Record not locked
31	Record number out of range
32	Dataset not found
33	Link tag error
34	Record already locked
35	Lock file not open
36	Error searching in lock file
37	Error writing to lock file
38	Error reading lock file
39	Process id error
40	Lock file outdated

41	Divide by zero
42	Out of bounds
43	Invalid object
44	Unimplemented instruction
45	Unresolved reference
46	Array reference error
47	Math domain error
48	Math range error
49	ADAPL file not found
50	FACS file not found
51	Print temp file error
52	Print write error
53	Print spool error
54	Obsolete function
55	Invalid FACS name
56	Invalid tag
57	Invalid CVT
58	No write access
59	No control format error
60	Text file error
61	Invalid occurrence
62	Tag list full
63	Data file write error
64	Operating system error

65	Multi-user error
66	Setup file error (Adlib.PBK)
67	Adlib.PBK outdated
68	Setup file (Adlib.PBK) not open
69	Object register error
70	Object register access error
71	Record list insert error
72	Hit list add error
73	Hit list remove error
74	No keywords left
75	Adlib.PRM file not open
76	Error in Adlib.PRM file
77	Invalid list position
78	Record overflow
79	Import file not open
80	No back-up files available
81	Dump file not open
82	File remove error
83	Duplicate record number
84	Missing record number
85	Error locking the linked record
86	Error initializing image processing
87	Error in the image file
88	Error loading image file

89	Error expanding image file
90	Write protection error
91	Duplication keyword
92	Invalid instruction
93	Error due to physical lock
94	Error searching index file
95	Error writing to index file
96	Error in index page
97	Error reading index page
98	Error in index definition
99	Error opening index definition
100	Index corrupt
101	Stack error
102	Error in temporary file
103	Record not selected
104	Record for synonym not found
105	Invalid path
106	Regular expression exceeds max. length
107	Too many parentheses
108	Parentheses do not balance
109	Error at the end of a regular expression
110	* or + error in regular expression
111	Syntax error in regular expression
112	Invalid range in square brackets

113	Square brackets do not balance
114	Syntax error in regular expression
115	Trailing \ error in regular expression
116	Regular expression corrupt
117	Error in regular expression
118	Error initializing image database
119	Return without gosub in ADAPL
120	Wrong image type
121	Narrower relation not mirrored
122	Imagebase create error
123	Imagebase not open error
124	Imagebase index not open error
125	Imagebase insert error
126	Imagebase search error
127	Image base write error
128	Image base read error
129	Image base delete error
130	Adeval procedure interrupted
131	Error in character set conversion table
132	Search file not found
133	Search file syntax error
134	Error writing pointerfile
135	Error opening pointerfile
136	Pointer file format error

137	Error reading pointerfile
138	Error during import or export job
139	Search interrupted
140	Error reading freelist
141	Error reading block in data file
142	Record header read error
143	Error in record header
144	Error expanding database
145	Error writing block to data file
146	Illegal input file format
147	Fieldname not found
148	Error writing to disk
149	Error while forcing linked record
150	Forcing not allowed
151	Error reading locked record
152	Logfile open error
153	Logfile write error
154	Circular internal link error
155	No preferred term
156	Field not empty
157	Duplicate term
158	Non-preferred term refers to preferred term
159	Preferred term has relation error
160	Term being used by other user

161-166	Self-issue loan desk error messages
167	No access error
168-170	Programming errors
171	Object file (.inf, .fmt, .pbk, .imp, .exp, etc.) corrupt
172	System file processed with newer version of software than version with which it was called
173	Recovery database path missing
174	Index on link reference field is incorrect or missing. (If your application uses feedback links, check that all forward reference fields in the primary databases that point to authority files have integer indexes.)
175	Import file has been made via logging, must be imported using recovery
176	Feedback record found using index on link reference field, but link reference not found in record itself
177	Index on linked field wrong type or missing
178	Linked field/link reference field cannot be removed from record retrieved via feedback links
179	Lock-file lock error (not yet used)
180	File cannot be found
181	Access to a file denied
182	File to be copied cannot be opened
183	File to be written to cannot be opened
184	Error reading a file
185	Error writing to a file
186	Cannot read or change date and time of a file
187	Cursor position changed in an adapt

188	User doesn't have access to a file (authorization failed)
189	General ODBC error. The error message provides more detailed information
190	Domain error: the domain name is too long
191	Left truncation has been set, but the indexes have not been adjusted accordingly
192	Preferred term relation hasn't been mirrored
193	Mandatory arguments haven't been submitted
194	Return date was not calculated
195	Location wasn't found
196	Search time-out
197	The registry can't be read
198	The registry can't be written to
199	The log file cannot be locked
200-250	Reserved for errors in AISODBC.dll (the ODBC driver)
251-300	Reserved for errors in Adlib Input/Output
301	Parse error: an error in the syntax of a search statement in the expert search system
302	Semantic factor relation hasn't been mirrored
303	MS_XML error
304	The record already exists
305	(no longer in use)
306	ADO error
307	Wwwopac error in http form: no request method has been submitted
308	Wwwopac error in http form: no content length has

	been submitted
309	Wwwopac error in http form: content length doesn't match
310	Wwwopac error in http form: no query data
311	Wwwopac error: general Adlib.DLL error, such as: this DLL not found; no access rights to the DLL; or licence file can't be found
312	Wwwopac error: semaphore error
313	Wwwopac error: error has already been handled
314	Wwwopac error: database header is missing
315	OAI error: wrong argument
316	OAI error: wrong resumption token
317	OAI error: wrong VERB
318	OAI error: cannot disseminate format
319	OAI error: OAI_ID doesn't exist
320	OAI error: no records found, or the set is empty or cannot be found
321	OAI error: no metadata formats present
322	OAI error: sets are not supported
323	OAI error: no ADMIN_EMAIL has been set
324	OAI error: no REPOSITORY name has been set
325	OAI error: no DELIMITER has been set
326	OAI error: no IDENTIFIER has been set
327	OAI error: no OAI_DIR has been set
328	OAI error: internal error
329	Gdiplus error

330	Not implemented
331	General error when exchanging data via GLUE with ADO
332	E-mail error. One of the possible causes could be that an adapl is trying to send e-mail with multiple attachments through SMTP: SMTP e-mailing allows only one attachment per e-mail.
333	Configuration error
334	Licence file error: licence invalid
335	Licence file error: licence has expired
336	Licence file error: licence cannot be found
337	Licence file error: cannot write to licence file
338	Licence file error: cannot read from licence file
339	Error in automatic numbering
340	Error in automatic numbering: counter file (.cnt) cannot be opened
342	Error in automatic numbering: counter file (.cnt) cannot be read
342	Error in automatic numbering: counter file (.cnt) cannot be written to
343	Error in automatic numbering: counter file error (.cnt)
344	Error in automatic numbering: lock on .cnt file causes error
345	ADLIBWEB error: general error within GLUE ADW
346	wrong lock ID
347	Error in internal link
348	Error with the type of the index file
349	Duplicate link reference tag

350	Duplicate destination tag
351	Link reference tag not filled
352	No linked data
353	Missing path DB separator: in a definition of a linked dataset in DBSETUP the + character cannot be found
354	String is too short
355	Import error: existing record for update tag not found.
357	C++ exception: a C++ exception is a programming error in the core software in Adlib, which may have widely differing causes. Previously, this type of programming error resulted in an "Illegal operation", causing adlwin.exe to shut down. This 357 error message however, is the result of a new (June 2005) built-in event handler that catches these illegal operations. Errors of this type should be reported to the Adlib Helpdesk.
<p>From Adlib 6.0, warnings and errors that Adlib generates, which usually appear in the screen of the user, are logged in the <i>Windows Event viewer (Control panel > Administrative tools/System management > Event viewer)</i>. The advantage of this is that if you have clicked an error message away, but want to look it up later once more, for instance because you need to provide the exact message to the Adlib Helpdesk, then in the <i>Event viewer</i> you can get an overview of errors that occurred in the past. By double-clicking an event, you open the full warning or error message. Do note that a warning usually does not indicate problems.</p>	

Windows error messages

Code	Cause
1001-1006	Internal Windows error
1007	The "busy" mouse cursor couldn't be loaded
1008	The help text file (adlibhlp) couldn't be opened
1009	The key entered for the help text doesn't occur in the help text file (adlib#.hlp or adlib#.adh)

1010	Screen definition file (.fmt) not found
1011	Required system field does not occur in screen definition
1012	Internal Windows error
1013	Printer data cannot be read
1014	The "welcome.bmp" file cannot be opened or displayed because of the PC's colour settings
1015	The printer cannot print on the selected paper format
1016	The printer cannot print in landscape/portrait format
1017-1024	Internal Windows error
-2147467259	Error during e-mailing. This error may be preceded by a Microsoft Office Outlook message stating that there might not be a default mail client or that the current mail client cannot fulfill the messaging request. The error is caused by the incompatibility of 64-bit versions of MS Outlook 2010 or higher with 32-bit MAPI calls from the 32-bit version of adlwin.exe. The error only occurs when the Windows MAPI is used to send e-mail from within Adlib (instead of using SMTP). The issue can be solved by replacing the 32-bit version of the Adlib executables by the 64-bit version.