Kofax Invoice Automation

Writing DCL programs

Version: 6.0.2

Date: 2019-05-10

KOFAX

Legal Notice

Table of Contents

Chapter 1

Introduction

Related documentation

A full set of the documentation for Kofax Invoice Automation can be found online here: https://docshield.kofax.com/Portal/Products/en_US/KIA/6.0.2-9nw1fk802p/KIA.htm

Getting help for Kofax products

Kofax regularly updates the Kofax Support site with the latest information about Kofax products.

To access some resources, you must have a valid Support Agreement with an authorized Kofax Reseller/Partner or with Kofax directly.

Use the tools that Kofax provides for researching and identifying issues. For example, use the Kofax Support site to search for answers about messages, keywords, and product issues. To access the Kofax Support page, go to www.kofax.com/support.

The Kofax Support page provides:

- Product information and release news

  Click a product family, select a product, and select a version number.
- Downloadable product documentation

  Click a product family, select a product, and click **Documentation**.
- Access to product knowledge bases

  Click **Knowledge Base**.
- Access to the Kofax Customer Portal (for eligible customers)

  Click **Account Management** and log in.

  To optimize your use of the portal, go to the Kofax Customer Portal login page and click the link to open the *Guide to the Kofax Support Portal*. This guide describes how to access the support site, what to do before contacting the support team, how to open a new case or view an open case, and what information to collect before opening a case.
- Access to support tools

  Click **Tools** and select the tool to use.
- Information about the support commitment for Kofax products

  Click **Support Details** and select **Kofax Support Commitment**.

Use these tools to find answers to questions that you have, to learn about new functionality, and to research possible solutions to current issues.

What is DCL?

DCL (Data Capture Language) programs can be used in Kofax Invoice Automation to improve extraction (interpretation). This document describes the concepts and details you need to write useful DCL expressions.

When to use DCL programs

It is not always better to use DCL expressions instead of standard keyword and format searches. DCL programs might be necessary if:

- You want to find a field without searching for a title. One example is if you want to capture a date that sometimes is written without a title. Another example is if the OCR interpretation of the keyword is poor but the field value OCR is good.

  A typical example is interpretation of numbers written on a Chinese document. For this to work, it is important that the format of the field you want to capture is unique. It must be possible to specify a format that only matches the value you want to capture.
- There is text between the title and the field you want to capture. Example: You want to capture the delivery address zip code.
- You want to capture several fields in one search. Example: You want to capture several address fields.
- The keyword is too short or not specific enough. Example: The keyword is "To". You must also include some additional nearby text to get a unique and correct match. This is possible with DCL.

Limitation

DCL only operates on the data from the full-page interpreter, that is, usually the interpretation engine for the printed text. That means that one can only use DCL to capture data when the data is machine printed.

Chapter 2

DCL language specification

DCL overview

DCL first searches for all possible matches of so-called atoms, which are small snippets, or short chunks of information, on the same line. Then it tries to put the atoms together according to a match expression.

To define the format of the atoms and match expressions, you use definitions.

Typically, more than one match is found for both atoms and match expression. To decide which one to choose, DCL gives every match a score according to these main criteria:

- The more characters, the higher the score.
- Every character that does not match the model results in fewer points.

This means that if no exact match with the atom or match expression is found, DCL will select an inexact match according to the same scoring criteria.

Basic DCL syntax

The easiest way to learn DCL is to look at a simple example. Let's say you want to find the invoice number on an invoice. In reality, you do not need a DCL in this case; a keyword and a format would suffice. However, here it can be used to demonstrate the basic concepts of DCL. Below are the six lines of our DCL program:

```
N [0123456789];
@InvNoKeyword 'Invoice number';
@InvNoData N{5-10};
Searchfor @InvNoKeyword ' ' @InvNoData;
match Searchfor;
output Field as Searchfor.@InvNoData;
```

See the following table for information about each line:

`N [0123456789]`	The numbers/characters that are expected in the results.
`@InvNoKeyword 'Invoice number';`	An atom. Their names must start with @. This one matches the text *Invoice number*
`@InvNoData N{5-10};`	This atom matches any 5 to 10 digit number. Here we use *N* that was defined above.
`Searchfor @InvNoKeyword ' ' @InvNoData;`	This defines `Searchfor` as `@InvNoKeyword` and `@InvNoData` separated by a space.

`match Searchfor;`	Each DCL program must contain exactly one *match definition*, which consists of `match` and a specification of what to match. Note that the previous line (the `Searchfor` definition) could be avoided by writing the match definition like this: `match @InvNoKeyword ' ' @InvNoData;`
`output Field as Searchfor.@InvNoData;`	The optional *output statement* (max. 1 per DCL program) tells what part of the match is to be outputted - in this example the invoice number without the keyword. The period (.) tells which part of the `Searchfor` to output.

As demonstrated by the example. a DCL program consists of:

- One or more *definitions* consisting of a *name* and a *format*. These names are arbitrary. How to specify the format is explained in the next section.

  The first four lines of the above example are these names + format definitions. They define `N`, `@InvNoKeyword`, `@InvNoData`, and `Searchfor`.

  Any format can include a previously defined name, so for example, the format `@InvoiceData` includes `N`, and the format of `Searchfor` includes `@InvNoKeyword` and `@InvNoData`.

  At least one of the definitions in a DCL program must be an *atom*. An *atom* is something you want DCL to search for - a short chunk of information on a single line. Its name starts with @ immediately followed by any combination of letters.
- After the definitions (including at least one atom) comes a *match expression* that specifies how the atoms appear in relation to each other.
- Optional: One or more *output declarations*. If the DCL is used to extract fields, the program must contain an output statement that defines which part of the match is to be outputted. See "Output declarations" on page 11
- Optional: *Background model*. See "The background model" on page 13

All DCL statements end with a semicolon (;).

Symbols and formats to use

	Symbol	DCL	Match examples
Strings	''	`@Keyword 'Invoice number';` `match @Keyword;`	Invoice number
		`@Keyword CaseIns'Invoice number';` `match @Keyword;`	Invoice number INVOICE Number Invoice Number
Subset	[ ]	`N [0123456789];`	1,2,3,4,5,6,7,8,9

	Symbol	DCL	Match examples

Specified range within a subset	-	`@AN [0-9A-Z];`
		`match @AN;`
		Note `N [0-9]` is the same as `N [0123456789]`
		The dash must be placed between numbers or explicit characters and within square brackets.
			A, B, C, ..., 1, 2, 3, ...

Ranges	{ }	`N [0-9];`
		`@TwoNumbers N{2};`
		`match @TwoNumbers;`
			12, 34, 87

		`N [0-9];`
		`@InvoiceNumber N{5-10};`
		`match @InvoiceNumber;`
			15678, 8739523, 9785634598

Optional (Zero to one)	?	`N [0-9];`
		`@Year '20' ? N{8};`
		`match @Year;`
			18, 2018

Zero to many	*	`N [0-9];`
		`@Numbers N*;`
		`match @Numbers;`
			<empty>
			3, 456, 290482649230

NOT (complement)	^	`NoNumber [^1234567890];`
		`@NoNumbers NoNumber*;`
		`match @NoNumbers;`
			Matches anything that does not contain any numbers.

Specific characters	\	`@Chars \a \b \c \d \e \f;` `match @Chars;`	a, b, c, d, e, f

Exclude characters from output	-	`@Year N{2} -('-' N{2} '-' N{2});`
		`match @Year;`
		`output field as @Year;`
			Finds 2018-04-02 but outputs only 2018

Any character except <space>	.	`@Anything .*;` `match @Anything;`	124135, df, 235li4u634 <empty> wetk23587thgl]]

Tab	tab	`@Keyword CaseIns' Date';`
		`@date N{2-4} '-' N{2} '-' N{2};`
		`SearchFor @Keyword tab @Date;`
		`match SearchFor;`
			Date 2018-04-02

Space	''	`@Keyword CaseIns'Date';`
		`@Date N{2-4} '-' N{2} '-' N{2};`
		`SearchFor @Keyword ' ' @Date;`
		`match SearchFor;`
			Date 2018-04-02

	Symbol	DCL	Match examples

New line	nl	`@Keyword CaseIns'Date';`
		`@Date N{2-4} '-' N{2} '-' N{2};`
		`SearchFor @Keyword nl @Date;`
		`match SearchFor;`
		Note that `nl` can only be used if the two atoms are left justified.
			Date
			2018-04-02

Check box		`@Box checkbox;` `match @Box;`	☐ and ☒
Or	|	`@POKeyword CaseIns'Order no' | CaseIns'PO number' | CaseIns'Purchase order';` `match @POKeyword;`	Order no PURCHASE ORDER PO Number
Grouping of expressions	( )	`@Keyword ('PO' |'Order') ('Number' | 'no');` `match @Keyword;`	PO Number PO no Order Number Order no
Comments	//	// Comments help your successors!	

Nearmove operators

These operators can be used to search for two atoms that are close to each other:

- `nearright` - unlike `tab` and `' '`, this does not require that the two atoms are written on the same line. It also allows characters between the two atoms.
- `nearleft`
- `neardown` - searches below without requiring that the two atoms are left justified.
- `nearup`
- `near`

Tips:

- `Tab`, `' '`, and `nl` (new line) provide the best matches if there are no characters between the two words and the words are on the same line. The nearmove operators allow more flexible positioning.
- If the field format is specific, you can use `neardown` and `nearright`. If the field formats are general, it is mostly better to use `tab`, `' '`, and `nl` to avoid incorrect matches.

  Example (general format, without nearmove operators):

```
@Keyword CaseIns'Invoice number';
InvNo [0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ\.-\/];
@InvoiceNumber InvNo{3-15};
SearchFor @Keyword (nl|tab|' ') @InvoiceNumber;
match SearchFor;
```

  Example (specific format, using nearmove operators):

```
@Keyword CaseIns'Bankaccount';
@BankNumber N{4} '-' N{4};
SearchFor @Keyword (nearright|neardown) @BankNumber;
```

```
match SearchFor;
```

Note Nearmove operators cannot be used in atoms.

Additional move operators

The following operators can also be used to connect unrelated parts of the document in the model. However, they are not as useful as the nearmove operators described above:

- `search` - search continues at any place on the invoice.
- `right` - right continues at any place on the invoice, prefers things to the right
- `downright` - etc.
- `down`
- `downleft`
- `left`
- `upleft`
- `up`
- `upright`

Note Move operators cannot be used in atoms.

More about atoms

- An atom must be on one line. It cannot span several lines.
- An atom cannot contain move operators (nearright, nearleft, neardown. etc.).
- Try to make the atoms as large as possible for best results.

Example of an unsuitable atom declaration:

```
@N [0123456789];
CustomerNumber @N{5-8};
match CustomerNumber;
```

Example of a suitable atom declaration:

```
N [1234567890];
@CustomerNumber N{5-8};
match @CustomerNumber;
```

Output declarations

The syntax for output is demonstrated by this example:

```
N [1234567890];
@CustomerNumber N{5-8};
match @CustomerNumber;
output Field as @CustomerNumber;
```

If the match expression only implicitly defines the value you want to output, you define the output like this:

```
N [0-9];
MomsRegHead 'Momsnr' | 'Vårt momsregnr' | CaseIns'VAT no'| 'Momsreg.nr';
OrgHead 'Org.nr' | 'Organisationsnr' | 'Org.nummer' | 'Organisationsnummer' |'reg nr';
Orgnr N{10}| (N{6} -'-' N{4});
@VATNo (('SE'|'se') Orgnr ('01')?)|Orgnr;
searchfor (OrgHead (neardown|tab|nearright) @VATNo)|(MomsRegHead (neardown|tab|
nearright) @VATNo);
match searchfor;
output suppliervatregistrationnumber as searchfor.@VATNo.Orgnr;
```

Note Replace Field by the Field type name as in the previous example (`suppliervatregistrationnumber`).

Chapter 3

Writing advanced DCL programs

The previous section gives you enough information to build simple DCL expressions that can be very powerful. however, for more complex problems, you need to understand the basics of how DCL works and understand the concept of a *background model*.

Understanding how DCL finds matches

DCL first searches for atom matches. In the second step, DCL tries to find which combination of all atom candidates that best matches the match expression. This means that if the correct value does not correspond to any of the atom candidates, you will not get the correct result. See the "Capturing parts of an address" example on page 20.

DCL uses a scoring strategy to determine the best match if several matches are found. The basic rules are:

- The more characters there are, the higher the score.
- A match with a specific format results in a higher score than a match with a more general format.
- Any character that does not match the format results in fewer points.

If there is no exact match, DCL finds the closest match according to these rules. If you want to avoid inexact matches, you may need to modify the *backfround model*.

It is possible to specify that the confidence score be calculated according to different criteria. Consider this example.

```
N [0123456789];
@InvNoKeyword 'Invoice number';
@InvNoData N{5-10};
Searchfor @InvNoKeyword ' ' @InvNoData;
match Searchfor;
output Field as Searchfor.@InvNoData;
```

In the output statement, scoring is based on `@InvNoData`. We could use the following output statement instead:

```
output Field as Searchfor.@InvNoData score Searchfor;
```

In this case, the confidence is based on the entire `Searchfor` definition, not just `@InvNoData`.

The background model

A match is considered to be found when the text matches the atom or match expression better than the *background model*.

DCL determines a score for the background model and the foreground model:

- More characters result in more points.
- A match with a more specific format results in more points.
- A deviation from the format results in fewer points.

The model that results in the highest score "wins".

You can apply a background model either to an atom or to the complete match expression.

If you do not specify a background model, the default background model, `(.|search)*`, is used. To avoid certain false matches, you can add them to the background model.

Example:

You want to extract a number that has exactly 8 digits but wish to avoid getting matches from longer numbers if no 8-digit number is found:

```
@Number N{8};
@Number background (N|N{9-12}|search)*;
match @Number;
```

As stated above, it is possible to specify a background model in connection with the match declaration:

```
N [0-9];
@Keyword 'Invoice number';
@InvNumber N{5-10};
BG (N|.|Search)*;
background BG;
match @Keyword neardown @InvNumber;
```

This causes DCL to regard numbers on te document as very common occurrences. See the following for more examples that include background models:

- The troubleshooting example on page 17.
- "Capturing parts of an address" example on page 20.
- "Capturing data from a Chinese invoice" example on page 21.

Tip In most cases you do not need to specify a background model. Always try to make the atoms and match expressions as effective as possible first. Only modify the background model when necessary, since it is difficult to predict the outcome.

Chapter 4

DCLs in Kofax Invoice Automation

Kofax Invoice Automation first tries to capture the fields with the keyword and format specified using the Kofax Invoice Automation Manager module. If the Interpret module fails to find a value, it does a second search based on the DCL expression.

Note If there are compilation errors in the DCL program, no warnings or error messages are displayed. The DCL program is simply ignored. The easiest way to check if the program compiles may be to test it in a Kofax Invoice Automation field.

Writing the DCL program

Write your DCL program in a text editor such as Notepad and save it with the `.DCL` extension.

In the output statement, use the Field type written in lowercase letters. For example, if the field type is SupplierVATRegistartionNumber, the output statement could look like the following:

```
output suppliervatregistrationnumber as searchfor.@VATNo.Orgnr;
```

Including the DCL expression in the invoice profile

Place the DCL file in the Country profile folder for which you want to use it (typically something like `C:\Users\Public\ReadSoft\INVOICES\GlobalPath\Country Profiles\SWE\6.2`.

Edit `Eiglobalextra.ini` in the same folder by adding the name of your DCL file in the [DCL] section. For example, if you have two DCL programs, `LineItem.DCL` and `OrgNr.DCL`, type the following in `Eiglobalextra.ini`:

```
[DCL]
DCLFile1=LineItem.DCL
DCLFile2=OrgNr.DCL
```

Chapter 5

Tips and Troubleshooting

If you are having trouble, see also "DCLs in Kofax Invoice Automation" on page 15

If the DCL program does not appear to be used

If the program does not appear to be used when invoices are interpreted:

- Ensure the DCL file is found in the country profile folder and that it is listed in `Eiglobalextra.ini`.
- Make sure you have written the field type name correctly with lowercase letters in the output line in the DCL program.
- Ensure the DCL program contains only characters that are present int he character set used by the country profile. (Using the Manager module, check **Invoice profile** > **Settings** > **Charecter set**.)
- Make sure there are no compilation errors in the program, for example:
  - Missing semicolon (;) at the end of each line
  - Missing ampersand (@) when referring to an atom
  - Unmatched apostrophe

The easiest way to check if the program compiles may be to test it in a Kofax Invoice Automation field.

If the DCL program does not result in the expected matches

If the DCL program is used but does not result in the desired matches, refer to the suggestions in the following sections.

Tips for optimal DCL programs

- Only use DCLs when it is necessary - when the simple model keyword + the field value is not enough.
- Make the DCLs as simple as possible.
- Only define a background model if necessary. Always start without an explicitly defined background model.
- Be aware that it is not possible to write a DCL program that correctly captures the field on *all* documents. Aim to capture the data in the most common scenarios.
- Preferably define about 1-4 atoms and 1-4 different alternatives how the atoms are placed relative to each other. If the DCL in too complex, it will become slow and may even give up searching all possible atom combinations.
- Make the atom formats as specific as possible.

Troubleshooting incorrect matches

If your DCL results in incorrect matches, make sure that the format is as specific as possible. For instance, if you know that the invoice numbers always contain a few numbers and maybe some capital letters, then use this:

```
ArtN [0-9\/\-A-Z];
@InvoiceNo (ArtN{0-4} N{3-4} ArtN{0-3});
```

... and not this:

```
ArtN [0-9\/\-A-Z];
@InvoiceNo (ArtN{4-10});
```

...since the latter will also match words written with capital letters.

If possible, use `' '`, `tab`, and `nl` instead of `nearright` and `neardown` between atoms, since `' '`, `tab`, and `nl` require that the atoms are placed directly to the left or directly below each other. Nearmove operators accept text between the atoms, and they accept when the atoms are not placed on the same line or directly below.

Modify the background model. The following example demonstrates how to avoid including the postal code and the address keyword when searching for address lines:

```
X [A-Za-z0-9,.];
N [0-9];
A [A-Za-z,.];

@Keyword 'Address';
Avoid 'Postcode:';
@Postcode ('GIR 0AA'|[A-PR-UWYZ]([0-9]{1-2}|([A-HK-Y][0-9]|[A-HK-Y][0-9]([0-9]|
[ABEHMNPRV-Y]))|[0-9][A-HJKPS-UW])[0-9][ABD-HJLNP-UW-Z]{2});
@Addressword N{0-3}(A|' '|tab){1-40};
@Addressword background (@Keyword|@Postcode|Avoid|.|search|tab|' ')*;

Address1 @Addressword neardown @Addressword neardown @Addressword;
Address2 @Addressword neardown @Addressword;
Searchfor (@Keyword (nearright|neardown) ((Address1|Address2) (neardown|tab|' ')
 @Postcode));

match Searchfor;

output Field as Searchfor.Address1;
output Field as Searchfor.Address2;
```

It is also useful to modify the backgound model when you search for a bank account number without including a keyword in the search, and you want to avoid capturing the customer's own account number.

```
N [0-9];
CustomerBanknumber '12345678901234567890123456';
@Banknumber -'nr'?N{26};
BG (CustomerBanknumber|N|search)*;
background BG;
match @Banknumber;
output supplieraccountnumber1 as @Banknumber;
```

The example above also contains `N` in the background model. This prevents matches if no 26-digit field is found on an invoice.

Still no matches - what now?

- Ensure that the word OCR of fields and keywords are correct - by rubberbanding the fields/keywords, for example.
- If your program contains more than one atom, try to first search for one atom at a time. If there are several values on your document that match the atom format, include the incorrect values in the atom background model. If you cannot find a match for one of the atoms, then check the following:
  - Is the field format correct?
  - Should it be more general or more specific? Normally the format should be as specific as possible. It may be necessary to accept not finding a match in certain special cases, in order to get good matches on *most* documents.
  - Do I need to modify the background model? Maybe include nearby text or numbers?

Chapter 6

Appendix: Examples

Capturing data without a keyword

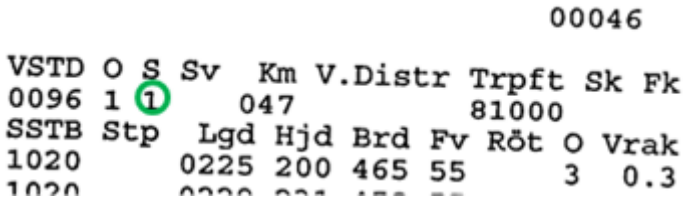
One of the most useful situations for using DCL is when you need to capture a value without a keyword. For this to work, the format must be unique so that it only matches the value you want to capture. Examples are account numbers, identification numbers, and company VAT registration numbers.

For a bank account number consisting of 6 numbers, a hyphen, and two additional numbers, the following DCL can be used:

```
N [0-9];
@AccountNumber N{6} '-' N{2};
match @AccountNumber;
output Field as @AccountNumber;
```

Capturing data with a short keyword

In this example, the keyword is S. Below the S, there is either a 1 or nothing. The 1 indicates if there is snow on the train rails.



When looking at the documents, we notice the following:

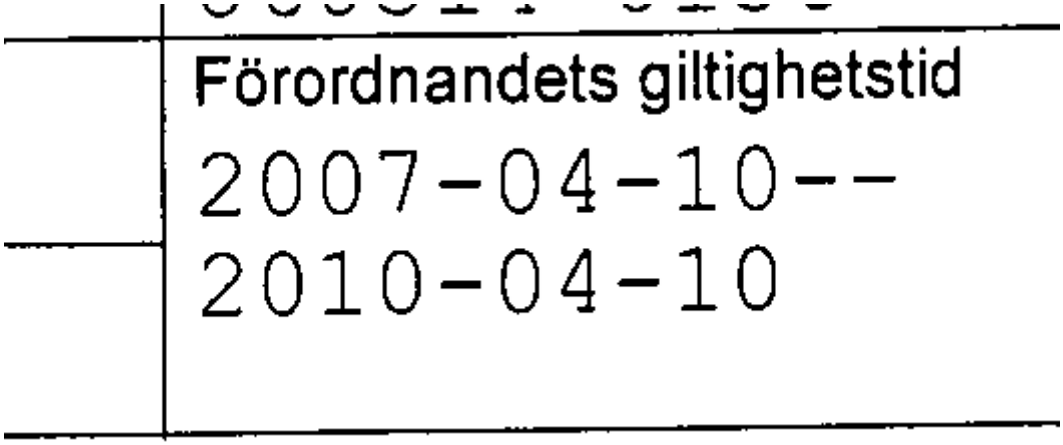
- We cannot search for an S above a 1, since DCL will more or less always find a match even if the area below S is clear.
- The row VSTD O S Sv Km V.Distr Trpft Sk Fk is always written that way.
- There is always a 3-digit number written below Km.
- The 1 is written directly below and is left justified with the S. Consequently, we use `nl` instead of `neardown`.
- We search for a format that finds a match both when a 1 is present and when it is not.

```
max alternatives=2;
N [0-9];
@Keyword 'S Sv Km V.Distr';
@Data '1';
```

```
@Km N{3};
Searchfor (@Keyword nl (@Data)? (' '|tab) @Km) ;
match Searchfor;
output Field as Searchfor.@Data score Searchfor;
```

One keyword, two dates

In this example, the keyword is S. Below the S, there is either a 1 or nothing. The 1 indicates if there is snow on the train rails.

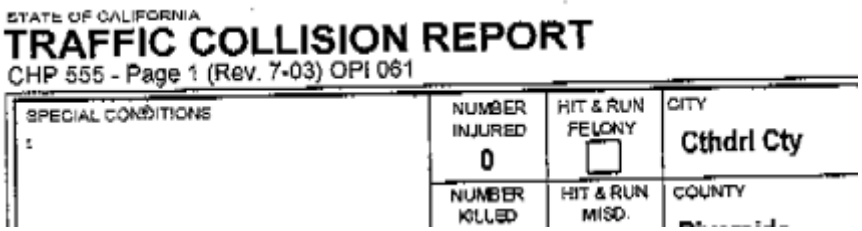


The first date field can be found using a regular keyword search. The second date can be found using the following DCL:

```
N [0-9];
@Keyword ‘Förordnandets giltighetstid’;
Date N{4} ‘-‘ N{2} ’-’ N{2};
@FromDate Date;
@ToDate Date;
SearchFor @Keyword neardown @FromDate ‘--‘? neardown @ToDate
match SearchFor;
output Field as SearchFor.@ToDate;
```

Check box capture

There is no built in-functionality for finding check boxes that are just below a certain keyword. However, you can have a DCL expression in a check box field which can be used to handle a check box such as the one shown here:


STATE OF CALIFORNIA
TRAFFIC COLLISION REPORT
CHP 555 - Page 1 (Rev. 7-03) OPI 061

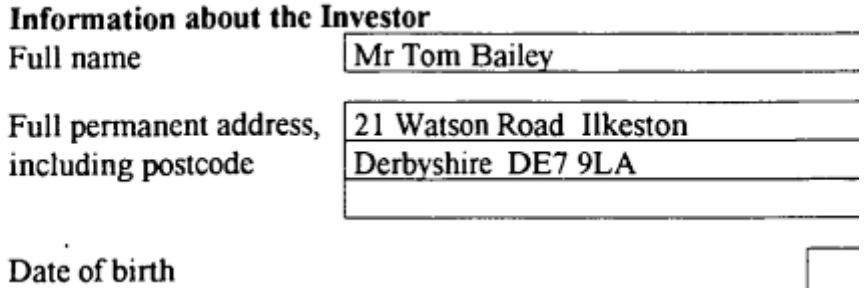
SPECIAL CONDITIONS	NUMBER INJURED	HIT & RUN FELONY	CITY
	0	☐	Cthdrl Cty
	NUMBER KILLED	HIT & RUN MISD.	COUNTY

The DCL expression below can be used in a check box field to find the check box shown above. Note that the expression states keywords that are to be found both above and below the check box. This may be needed if there are other check boxes below the one we want to find.

```
max alternatives=2;
@Data checkbox;
@HitAndRun 'HIT & RUN';
@Felony 'FELONY';
Searchfor @HitAndRun neardown @Felony neardown @Data neardown @HitAndRun;
match Searchfor;
output Field as Searchfor.@Data score Searchfor;
```

Capturing parts of an address

We want to capture the address, excluding the postcode, from this kind of document:


Information about the Investor

Full name	Mr Tom Bailey
Full permanent address, including postcode	21 Watson Road Ilkeston Derbyshire DE7 9LA
Date of birth	

Note that the format of the address is very general. It can contain letters, numbers, spaces and tabs.

If we define an atom `@Addressword` with the format `(X|' '|tab){1-60}`, DCL will find the following atom matches:

- Information about the Investor
- Full name
- Mr Tom Bailey
- Full permanent address, 21 Watson Road Ilkeston
- Including the postcode Derbyshire DE7 9LA

If we define another atom `@Keyword` with the format "Full permanent address" and define the following:

```
match @Keyword (tab|’ ‘) @Addressword;
output Field as @Addressword;
```

...it will not match "21 Watson Road Ilkeston", since DCL has not identified this as one of the atom matches.

The solution is to add "Full name", "Full permanent address", "including postcode", and the postcode format to the `@Addressword` background model. Then, DCL will find the following atom matches for `@Addressword`:

- Information about the Investor
- Mr Tom Bailey
- 21 Watson Road Ilkeston
- Derbyshire

Now this:

```
match @Keyword (tab|’ ‘) @Addressword;
output Field as @Addressword;
```

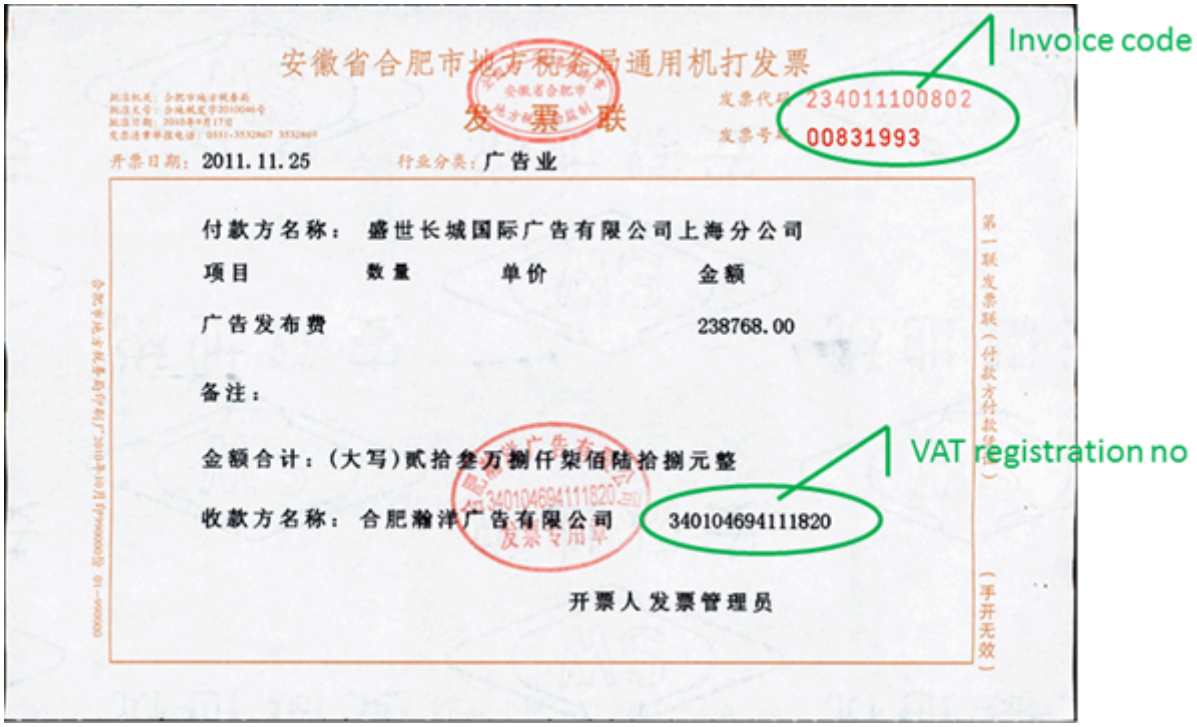
...will match 21 Watson Road Ilkeston in the way we want:

```
X [A-Za-z0-9,.];
N [0-9];
A [A-Za-z,.];
@Keyword 'Full permanent address';
Avoid 'including postcode'|'Address, including';
@Postcode ('GIR 0AA'|[A-PR-UWYZ]([0-9]{1-2}|([A-HK-Y][0-9]|[A-HK-Y][0-9]([0-9]|
[ABEHMNPRV-Y]))|[0-9][A-HJKPS-UW])[0-9][ABD-HJLNP-UW-Z]{2});
@Addressword N{0-3}(A|' '|tab){1-40};
@Addressword background (@Keyword|@Postcode|Avoid|search|tab|' ')*;
Address @Addressword neardown @Addressword;
Searchfor (@Keyword (nearright|neardown) ((Address) (nl|tab|' ') @Postcode));
match Searchfor;
output From as Searchfor.Address;
```

Capturing data from a Chinese invoice

Capturing data from Chinese invoices is difficult due to several reasons: The quality of the original invoices is poor, text is written with different colors for different parts of the invoice, and the Chinese characters contain very fine details. Interpretation performance is therefore poor if only the standard keyword + format is used to locate the field values.

Since numbers are more accurately extracted than characters, we make use of the fact that it is possible to define a format that uniquely identifies the field.



Invoice code

On the invoices we are looking at, we notice the invoice code (in the sample image 234011100802):

- Always starts with "2"
- Always has 12 digits
- Always has the 8-digit invoice number printed below

This can be used to uniquely capture the invoice code. We can also search for the invoice code keyword and format.

We define three atoms:

- Keyword
- Invoice code
- Invoice number

...and search for two different layouts:

- keyword + invoice code
- invoice code above invoice number

The DCL program can then be written as follows:

```
max alternatives=3;
N [0-9] -' '?;
Nback [0-9.,] -' '?;
@InvCode '2' N{11};
@InvCode background (Nback|search|tab|' '|N{13})*;
```

```
@InvNo N{8};
@InvNo background (Nback|search|tab|' ')*;
@KeyWord '发票代码' | '代码' | CaseIns'INVOICE CODE';
Layout1 @KeyWord (tab | ' ') @InvCode;
Layout2 @InvCode neardown @InvNo;
document (Layout1 | Layout2 );
BG (Nback|search|tab|' ')*;
background BG;
match document;
output Field as document.Layout1.@InvCode score document.Layout1;
output Field as document.Layout2.@InvCode score document.Layout2;
```

Note 1: On these invoices, the distance between digits is large, so the OCR engine interprets a space between them. This is solved by defining N as:

```
N [0-9] -' '?;
```

Note 2: We specify a background model to avoid matches with numbers that are not precisely 12 and 8 digits.

Note 3: When determining which of all possible matches to choose, we want to account for the complete layouts. Therefore, we explicitly specify how the scoring should be calculated:

```
output Field as document.Layout1.@InvCode score document.Layout1;
```

If we omit the score part in the example above, the scoring will only take into account how well the invoice code matches the invoice code format.

VAT registration number

When inspecting Chinese invoices , we notice that the tax registration number:

- Consists of 15 digits (sometimes including 5 initial zeros and an X at the end)
- Is sometimes written next to a keyword
- Is sometimes only written in a stamp without any keyword
- The quality of the printed value is often poor quality such that the OCR of the number is incorrect

```
max alternatives=3;
N [0-9] -' '?;
Nback [0-9.,:] -' '?;
Prefix -(':'|'.'|tab|' '|'00000');
@KW '税号' | '税务登记号' | '税务登记号码' | '收款方识别号' | '纳税人识别号' | '收款方纳税识别号'
 |
 '收款单位税号' | '身份证号' | '收款方编码' | 'TAX' | 'TAX REGISTRY NO';
@TaxRegNo Prefix? (N{15}|(N{14} 'X') ) -(tab|' ')?;
@TaxRegNo background (Nback|tab|search|' '|'111111111111111')*;
SearchFor (@KW nearright @TaxRegNo) | @TaxRegNo;
match SearchFor;
output Field as SearchFor.@TaxRegNo score SearchFor;
```

Note We specify a background model to avoid matches with numbers that are not exactly 15 digits. We also include a period (.) and a comma (,) in the background model to avoid false matches with amount fields. Barcodes were sometimes incorrectly interpreted as 111111111111111, and were also added to the background model as a consequence.

Free-text search without keyword

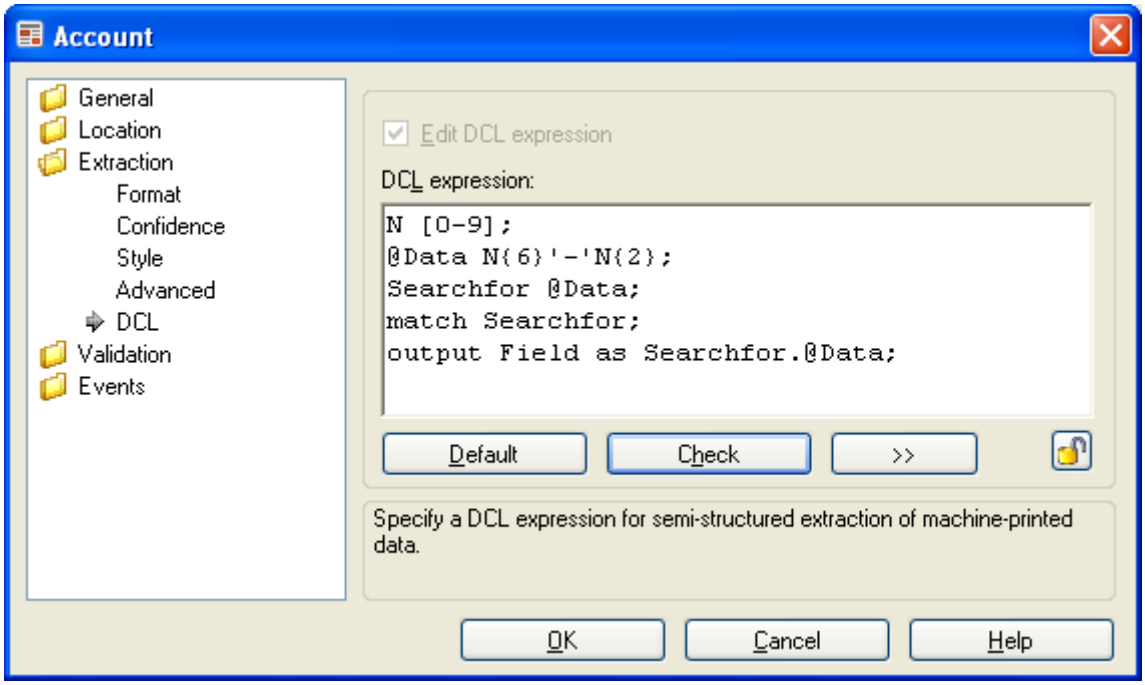
Sometimes you cannot find proper keywords for the data that you want to extract because the keywords are unknown or are not unique within the document, such as:

- ...my bank account number is 123456-34...
- ...I have an account with your bank, 123456-34...
- ...please use my account, 123456-34,...

If the data that you want to find has a format that does not appear on other places in the document, it can be found and extracted using only that format. This can be done using a DCL expression in a character field, date field or amount field.

Character field

The DCL expression is placed in the DCL part of the character field. The ordinary keyword location methods should be removed so that there is only one extraction method.



The DCL expression used to locate the bank account number:

```
N [0-9];
@Data N{6}'-'N{2};
Searchfor @Data;
match Searchfor;
output Field as Searchfor.@Data;
```

Keyword is not unique

Extraction results can sometimes be improved greatly by using DCL.

If we want to extract the name of the receiver in mails like the one below, we can use the keyword 'To:'. However, "To" is a very short keyword and a very common word, so we do not know if there are any similar words in the surrounding area.

In this case we can improve extraction performance by using a DCL expression that not only defines the keyword 'To:' and its relation to the data to extract but also the spatial relationship to surrounding keywords like "Sent:" and "Subject:".

From: Udo Diss
Sent: September 21 2009 11:23
To: Ivanna Lurn
Subject: It is good to use DCL in some cases

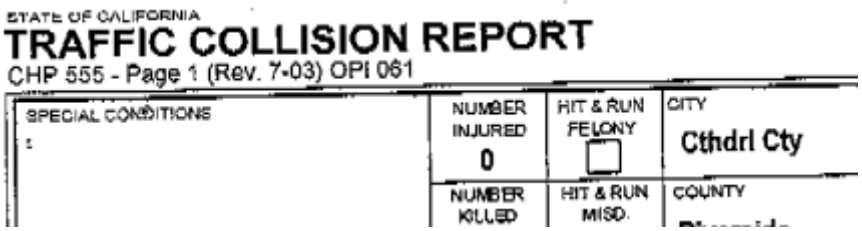
Hello,
The purpose of this email is to show you that DCL can improve extraction results.

This DCL expression finds the name of the receiver by defining the relationship to the keywords "Sent:", "To", and "Subject:".

```
A [A-Z\-];
a [a-z\-];
@Name A a{2-30}' 'A a{2-30};
@Sent 'Sent:';
@To 'To:';
@Subject 'Subject:';
Searchfor @Sent neardown (@To tab @Name) neardown @Subject;
match Searchfor;
output Field as Searchfor.@Name;
```

DCL and check box fields

There is no built-in functionality for finding check boxes that are just below a certain keyword. However, you can have a DCL expression in a check box field that can be used to handle a check box like the one shown in the following example.


STATE OF CALIFORNIA
TRAFFIC COLLISION REPORT
CHP 555 - Page 1 (Rev. 7-03) OPI 061

SPECIAL CONDITIONS	NUMBER INJURED	HIT & RUN FELONY	CITY
1	0	☐	Cthdrl Cty
	NUMBER KILLED	HIT & RUN MISD.	COUNTY

This DCL expression can be used in a check box field to find the check box in the example shown above. Note that the expression states what keywords are to be found both above and below the check box. This may be needed if there are other check boxes below the one we want to find.

```
beamsearchwidth=0;
greedylimit=500000;
max alternatives=2;
@Data checkbox;
@HitAndRun 'HIT & RUN';
@Felony 'FELONY';
Searchfor @HitAndRun neardown @Felony neardown @Data neardown @HitAndRun;
match Searchfor;
output Field as Searchfor.@Data score Searchfor;
```