The Unscrambler Appendices: Technical References

This document references the more technical features of the software: file formats and composition, compatibility with instruments, user-defined transformations.

Contents

Unscrambler Files

This chapter introduces the various types of files that either build up The Unscrambler (runtime files) or are generated by The Unscrambler to store your data, transformations and results.

Unscrambler Runtime Files

This is a list of all the files which are needed to run The Unscrambler, or are generated run time by The Unscrambler, except for the dll-files which will be placed in the Windows\System directory.

Main Files

These are the “core” of the program: executable, help file, and a few more.

Main Unscrambler files

File name	Purpose	Note
Unscramb.exe	Executable	*
Unscramb.hlp	Help file	*
Unscramb.cnt	Help context file	*
Readme.txt	Installation hints. Changes/fixes since last release. Known bugs, etc	*

* Delivered with the program

System Files

System files are necessary for the Unscrambler to keep track of your specific configuration.

Unscrambler system files

File name	Purpose	Note
Unscramb.sys	System file with license information etc.	*
Userdefs.<user initials>	User Defaults (system file)	**
$login$.<user initials>	Signature files for users logged in to The Unscrambler.	** *

* Delivered with the program

** Each user has his/her own User Defaults file

*** Exists only while running the program

Example Files

These files contain example data, useful to run the Tutorials.

Unscrambler example files

File type	Purpose	Note
Data tables	Example data tables	*
Result files	Example result files	*

*Delivered with the program

File Naming Conventions

For each new data and result file about to be saved, The Unscrambler generates a default file name which the user may override. The table below lists the default file names:

Default file names

File type	Default Name
Data tables	**DATA**<data table number>
Result files	**RESULT**<result file number>

File Extensions Used by The Unscrambler

File Extensions of Unscrambler Files

The Unscrambler uses the file extensions listed below to distinguish between different files types.

Unscrambler file extensions

Unscrambler Files:	File Extension
Non-designed raw data	.00D
Fractional factorial design	.01D
Full factorial design	.02D
Combined design	.03D
Central Composite design	.04D
Plackett-Burman design	.05D
Box-Behnken design	.06D
D-optimal design	.07D
Statistics	.10D
PCA	.11D
Analysis of Effects	.20D
Response Surface	.21D
Prediction	.30D
Classification	.31D
MLR	.40D
PLS1	.41D
PLS2	.42D
PCR	.43D
Three-way PLS	.44D
MSC	.50D
Lattice design (mixtures)	.60D
Centroid design (mixtures)	.61D
Axial design (mixtures)	.62D
D-optimal mixture design	.63D
3-D data table	.70D

Each of the files above (*.??D) may have the following corresponding additional files:

*.??L Log file

*.??P Preference file (settings for the file when it closes)

*.??T Notes file

*.??W Warnings file

Note:

Each Unscrambler data table or result file consists of 4 or 5 physical files. If you want to transfer data to another place using the Windows Explorer, make sure that you copy all the associated physical files!

The log and notes files are plain ASCII files which you may open and take look at using a text editor.

The Unscrambler ASCII-MOD file is saved with the extension .AMO.

Note:

Do not change the file extensions The Unscrambler uses. You may find that you cannot access the files from within The Unscrambler when the file extension is changed.

File Extensions of External Files

The Unscrambler uses default file name extensions. If you want to import an external file, it assumes that it has the extensions listed below. Files that do not use these extensions will not be displayed in the file dialog. Entering other file extensions is not allowed.

File extensions, external files

Files:	File Extensions
ASCII	.INP, .DAT, .TXT, and .ASC
Old Unscrambler	.UNS, .UNM, .UNP, and .CLA
Excel	.XLS
Lotus	.WK3 and .WK4
JCAMP	.JDX
APC	.HDR
NSAS	.DA
Tracker	.CAL
Grams	.SPC, .CFL
Matlab	.M
Guided Wave CLASS-PA and SpectrOn	.ASC, .SCN and .AUTOSCAN
Indico	.ASD, .nnnn (any number)
Hitachi F3D	.F3D

File Structures

The Unscrambler can save files in different formats. Some of the file formats are described hereafter.

ASCII-MOD File Structure

An ASCII-MOD file contains all information necessary for prediction and classification.

The ASCII-MOD file is an easy-to-read ASCII file. The table below lists the matrices which are found in the ASCII-MOD file, depending of type of ASCII-MOD file and type of model. When generating an ASCII-MOD file, you can choose between “MINI” and “FULL” storage. Matrices stored under these options are indicated with ‘x’ in the table.

ASCII-MOD file matrices

Matrix name	MINI	FULL PCA	FULL Regr.	Rows	Columns
B	x		x	PC (1-a)	X-var (1-x)
B0	x		x	PC (1-a)	1 row
xWeight		x	x	1 row	X-var (1-x)
yWeight			x	1 row	Y-var (1-y)
xCent		x	x	1 row	X-var (1-x)
yCent			x	1 row	Y-var (1-y)
ResXValTot		x	x	PC (0-a)	
ResXCalVar		x	x	PC (0-a)	X-var (1-x)
ResXValVar		x	x	PC (0-a)	X-var (1-x)
ResYValVar			x	PC (0-a)	Y-var (1-y)
ResXCalSamp		x	x	PC (0-a)	Samp (1-i)
Pax		x	x	PC (1-a)	X-var (1-x)
Wax		x	x	PC (1-a)	X-var (1-x)
Qay			x	PC (1-a)	Y-var (1-y)
SquSum		x	x	***)**	PC (1-a)
HiCalMean			x	PC (1-a)	1 row
ExtraVal			x	1 row	****)**
RMSECal			x	PC (1-a)	Y-var (1-y)
TaiCalSDev		x	x	PC (1-a)	1 row
xCalMean		x	x	1 row	X-var (1-x)
xCalSDev		x	x	1 row	X-var (1-x)
xCal		x	x	1 row	X-var (1-x)
yCalMean			x	1 row	Y-var (1-y)
yCalSDev			x	1 row	Y-var (1-y)
yCal		**)**	x	1 row	Y-var (1-y)

Table of result matrices:

***)** SquSumT, SquSumW, SquSumP, SquSumQ, MinTai, MaxTai

****)** RMSEP, SEP, Bias, Slope, Offset, Corr, SEPcorr, ICM-Slope, ICM-Offset

Note: The contents of the columns “Rows” and “Columns” shows the contents of the ASCII-MOD file, not the contents of the matrices in the main model file.

Example of an ASCII-MOD File

```
TYPE=FULL                (MINI,FULL)
VERSION=1
MODELNAME=F:\U\EX\DATA\TUTBPCA.11D
MODELDATE=10/27/95 11:41:13
CREATOR=Helge Iversen
METHOD=PCA               (PCA,PCR,PLS1,PLS2)
CALDATA=F:\U\EX\DATA\TUTB.00D
SAMPLES=28
XVARS=16
YVARS=0
VALIDATION=LEVCORR       (NONE,LEVCORR,TESTSET,CROSS)
COMPONENTS=2
SUGGESTED=2
CENTERING=YES                    (YES,NO)
CALSAMPLES=28
TESTSAMPLES=28
NUMCVS=0
NUMTRANS=2
TRD:DNO ,,,,,, complete transformation string
TRD:DSG ,,,,,,,complete transformation string
NUMINSTRPAR=1
##GAIN=5.2
MATRICES=13
"xWeight"                (Name of 13 matrices)
"xCent"
"ResXValTot"
"ResXCalVar"
"ResXValVar"
"ResXCalSamp"
"Pax"
"Wax"
"SquSum"
"TaiCalSDev"
"xCalMean"
"xCalSDev"
"xCal"
%XvarNames
"Xvar1"  "Xvar2"  "Xvar3"  "Xvar4"
"Xvar5"  "Xvar6"  "Xvar7"  "Xvar8"
"Xvar9"  "Xvar10"  "Xvar11"  "Xvar12"
"Xvar13"  "Xvar14"  "Xvar15"  "Xvar16"
%xWeight              1     16
.1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01
.1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01
.1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01 .1000000E+01
.1000000E+01
%xCent                1     16
.1677847E+01 .2258536E+01 .2231011E+01 .2404268E+01 .2179311E+01
.2470489E+01 .2079168E+01 .1734536E+01 .1475164E+01 .1480657E+01
.1644097E+01 .1805900E+01 .1980229E+01 .1795443E+01 .1622796E+01
.1497418E+01
,,,
,,,etc.…
```

Description of Fields

Description of fields

Field	Description
TYPE	(MINI,FULL) MINI gives “Prediction Light” only
VERSION	Increases by one for each changes of the file format after release.
MODELNAME	Name of model
MODELDATE	Date for creation of the model (not the ASCII-MOD file)
CREATOR	Name of the user who made the model (not the ASCII-MOD file)
METHOD	Calibration method (PLS1, PLS2, PCR, PCA)
CALDATA	Name of data set used to make the model
SAMPLES	Number of samples used when making the model
XVARS	Number of X variables used when making the model
YVARS	Number of Y variables used when making the model
VALIDATION	(TEST,LEV,CROSS)
COMPONENTS	Number of components present in the ASCII-MOD file
SUGGESTED	Suggested number of components to use (may not be on the ASCII-MOD file)
CENTERING	(YES,NO)
CALSAMPLES	Number of calibration samples
TESTSAMPLES	Number of Test samples
NUMCVS	Number of Cross Validation Segments
NUMTRANS	Number of transformation strings
INSTRUMENT PARAM.	See below
TRANSFORMATIONS	Number of transformations
MATRICES	Number of matrices on this file. One name for each matrix follows below

Transformation String

One line for each transformation. The format of the line will depend on type of transformation. If a transformation needs more data which is the case for MSC, this extra data will be stored as matrices at the end of the file. References to these matrices can be done by names.

Examples

A transformation named TRANS using one parameter could look like this:

TRANS:TEMP=38.8;

A MSC transformation may look something like this:

```
MSC:VARS=19,SAMPS=23,MEAN="ResultMatrix18",TOT=" ResultMatrix19";
```

(see Abbreviations Used in Log Files for an explanations of the abbreviations)

Transformation strings may also contain error status which is the case when the MSC-base have been deleted from file before making the ASCII-MOD file.

Storage of Matrices

Each matrix starts with a header as in this example:

```
%Pax 10 155
```

Telling: Matrix name is **Pax** the matrix has the dimension **10** rows and **155** columns. From the next line the data elements will follow in the following sequence:

```
Pax(1,1) Pax(1,2)     Pax(1,3)     ,      ,      Pax(1,7)
Pax(1,8) Pax(1,9)     ,            ,      ,      ,
Pax(1,xvars-1) Pax(1,xvars)
Pax(1,2) Pax(2,2)     Pax(2,3)     ,      ,      ,
, ,
Pax(comp,1) Pax(comp,2),     ,      , Pax(comp,xvars)
```

A **missing** value will simply be written as the character **m**.

Unscrambler ASCII File Format

Previous Unscrambler versions use conventional ASCII files of the type generated by text editors and includes a file head. This format can be used for other ASCII-files as well, i.e. you can use a text editor to edit your file head to be equal to the one described below.

The Unscrambler ASCII file format is:

```
% <L> <C> mmmmmmmttttttttttttttttttt.....ttttttttt
<m(1,1)>  <m(1,2)> .. <m(1,C)>
<m(2,1)>    .....      .....
<m(L,1)>    .....    <m(L,C)>
#L
name1  name2  name3  ........  nameL
#C
```

name1 name2 name3 ..... nameC

Description of the format

%

Start of matrix head. Must be at the first leftmost position.

<L>

An integer value representing the number of lines in the matrix.

<C>

An integer value representing the number of columns in the matrix.

mmmmmm (Optional)

A six character matrix name.

tt..tt (Optional)

A 60 character matrix text.

<m(i,j)>

A matrix element (real or integer).The character 'm' or the value - 0.9973E+24 denotes a missing value.

#L (Optional)

Start of line names (Object names). The character # must be at the first position.

#C (Optional)

Start of column names (Variable names). The character # has to be at the first position.

name?

An object or variable name. Maximum 8 characters.

Delimiters

Space, comma and all characters with ASCII value between 0 and 32.

Line size

Max 130 characters, the rest are skipped.

If any sample or variable name starts with '#' or '%' and this character happens to be in the first column, it is treated as an error.

Unscrambler 5 Data Files

The information below is only interesting if you need to read Unscrambler 5 files into your own application software.

Format: Binary

Access: Direct

Record length: 512 bytes

- The maximum number of matrices on one file is 11.
- The two first blocks contains file and matrix heads.
- The file head contains a comment field of 256 characters.

Each matrix head has two name indices which point at two name array start addresses; one for line names and one for column names.

No names are available if indices are ZERO. Pointers are accessed via indices because several matrices may use the same names. Syntax used below:

{min,max}: The value of the field have to be in the range min,max
I2 : Type INTEGER*2
I2(7) : Type INTEGER*2
TIME(7) : (1/100 sec, sec, min, Hour, day, month, year)
I4 : Type INTEGER*4
L2 : Type LOGICAL*2
Ax : Type CHARACTER*x
Hxx : Matrix head of xx bytes

All data/name positions are BytePos/4. For instance, the first element of block 3 is position 257. The matrices are stored linewise or columnwise with each element represented as a 4-byte real.

Block 1

Byte Position	Type	Description
1- 2	I2	Number of matrices on file {0,11}
3- 4	I2	Number of name arrays {0,22}
5-18	I2(7)	Date/Time of creation
19-32	I2(7)	Date/Time of last update
33-42	A10	File system identification: VALUE = "22oct90ExC"
43-46	I4	End Of File position (BytePos/4)
47-48	L2	Write Protected flag .TRUE. if write protected
49-50	I2	File type (see below)
51-160	???	UnUsed
161-164	I4	Start position name array 1 (BytePos/4)
165-168	I4	Start position name array 2 (BytePos/4)
. . .		
245-248	I4	Start position name array 22 (BytePos/4)
249-256	8	UnUsed
257-512	A256	Buffer for comments (the last four characters are not shown in The Unscrambler's comments field)

Block 2

Byte Position	Type	Description
1-44	H44	Matrix head no 1
45-88	H44	Matrix head no 2
. . .		
441-484	H44	Matrix head no 11
485-512	28	UnUsed

Matrix head

rel.	Byte pos.	Type Description
1- 6	A6	Matrix name
7-14	8	Unused
15-16	I2	Number of lines {1,32765}
17-18	I2	Number of columns {1,32765}
19-20	I2	First line number {0,1} used by plot module only
21-22	I2	First column number {0,1} used by plot module only
23-24	I2	Matrix type 0=data,1=code
25-30	6	Unused
31-32	I2	Line name ID (index to name array adr) {0,,22}
33-34	I2	Column names ID (index to name array adr) {0,,22}
35-36	I2	Storage Linewise/Columnwise (see below)
37-38	I2	Line types (see below)
39-40	I2	Column types (see below)
41-44	I4	Data start position (BytePos/4)

File types:
9372 : General data file
12791 : Output from calibration
29137 : Output from prediction
11911 : Design file
17964 : Classification file
Other file types will not be accepted.

Storage:
76 Linewise: First line 1, then 2 etc.
67 Columnwise: First column 1, then 2 etc.

If value of storage is not 67 or 76 result of read/write will be garbage.

Line/Column types (valid for matrices, generated during calibration and prediction)

0 : Undefined
1 : Objects
2 : Variables
3 : Principal components

4 : Principal components
5 : Models
6 : X-variables
7 : Y-variables
8 : Statistics

Unscrambler 5 Model Files

The table below gives a description of the Unscrambler 5 system matrices for calibration, classification and prediction files.

Unscrambler 5 system matrices

Matrix name	Dimension (L*C)	Calibration file	Prediction file
System	1 * 40	C00mmmmv.UNM	pppmmmmv.UNP
ObjSys	(NOB+1)/2 * 1	C00mmmmv.UNM	
Outlie	4 * x	C00mmmmv.UNM	pppmmmmv.UNP

The above dimensions are measures in elements of 4 bytes each.

Syntax used for the Type field below:

I2: Type INTEGER*2 (2 bytes)

R4: Type REAL*4 (4 bytes)

A12: Type Text (12 characters)

The "System" matrix

Byte pos.	Type	Name	Description
1-2	I2	NXV	Number of X-variables in data set
3-4	I2	NYV	Number of Y-variables in data set
5-6	I2	NOB	Number of objects in data set
7-8	I2	NOISE	Start noise, 1=Yes 2=No
9-10	I2	OPTF	The optimal number of factors
11-12	I2	NFAC	Number of factors
13-14	I2	VMET	Validation method, 1=Leverage Correction 2=Test set 3=Cross validation
15-16	I2	CMET	Calibration method, 1=PCA 2=PCR 3=PLS1 4=PLS2
17-18	I2	SSEL	Segment selection, 1=Random 2=Systematic

Byte pos.	Type	Name	Description
19-20	I2	TSEL	Test set selection, 1=Random 2=Manual
21-22	I2	ODET	Outlier detection, 1=On 2=Off
23-24	I2	LDET	Leverage detection, 1=On 2=Off
25-26	I2	NTS	Number of test objects
27-28	I2	NREMO	Number of removed objects
29-30	I2	NCVAL	Number of cross validation segments
31-32	I2	YPLS1	Y-variable number used during PLS1
33-36	R4	OLIM	Outlier detection limit
37-40	R4	LLIM	Leverage detection limit
41-42	I2	CSTAT	Calibration status, 0=Ok
43-54	A12	CDATE	Calibration date (e.g. "Dec 08 1992")
55-56	I2	WSTAT	Status warning buffer, 0=Ok 1=Full
57-58	I2	WNOUT	Number of outlier warnings
59-60	I2	WNLEV	Number of leverage warnings
61-62	I2	SRES	Residual. number of factors, 1=None 2=Optimal 3=All
63-64	I2	SB	B-coefficient. number of factors, 1=N 2=O 3=A
65-66	I2	SBW	BW-coefficient. number of factors, 1=N 2=O 3=A
67-68	I2	SPRE	Y-predicted. number of factors, 1=N 2=O 3=A
69-70	I2	SWGT	Weighted data set, 1=Yes 2=No
71-72	I2	SCVST	CVS: Total Y-variance, 1=Yes 2=No
73-74	I2	SCVSY	CVS: Y predicted, 1=Yes 2=No
75-76	I2	SCVSEVA	CVS: Extra validation, 1=Yes. 2=No ******)**
77-78	I2	BDIA	B-coefficient diagnoses, 1=Yes 2=No ***)**
79-80	I2	BWDIA	BW-coefficient diagnoses, 1=Yes 2=No ***)**
81-92	A12	XF	X-matrix file name
93-98	A6	XM	X-matrix matrix name
99-100	I2	MCTR	Model center, 1=mean(var). 2=origo *******)**
101-112	A12	YF	Y-matrix file name
113-118	A6	YM	Y-matrix matrix name
119-120			UNUSED
121-132	A12	PF	File name, X-matrix for prediction ****)**
133-138	A6	PM	Matrix name, X-matrix for prediction ****)**
139-140	I2	NPFAC	Number of factors in prediction ****)**
141-142	I2	NPNOB	Number of objects in prediction matrix ****)**
143-144	I2	YREF	Predict with reference, 1=Yes 0=No ****)**
145-148			UNUSED
149-156	A8	YPLS1N	Name for Y-variable used during PLS1
157-158		DESIGN	1=Design model. 2=Ordinary model *****)**
159-160	I2	VER	System matrix version number

***)** Used only when VER > 0

****)** Stored only in the prediction file

*****)** Used only when VER > 1

******)** Valid only when VER > 2

*******)** Valid only when VER > 3

The "Objsys" matrix

Byte pos.	Type	Name	Description
1-2	I2	Object_1	Object control code
3-4	I2	Object_2	Object control code
...	...	...	...
...	...	...	...
x-x	I2	Object_NOB	Object control code

Values of control codes for "Objsys" matrix

Value of control code	Description
-4	Object is a frozen calibration object (CVS)
-1	Object belongs to test set
0	Object is removed
n > 0	Object belongs to cross validation segment number n.
1	Object belongs to calibration set. or CVS seg. no 1.

Outlie

Linked list of outlier warnings, rather complex.

JCAMP-DX File Format

This format is used by many spectroscopy instrument vendors, eg. Bran+Luebbe (IDAS/Infralyzer), NIRSystems (NSAS), Perkin Elmer, Galactic (Grams), etc.

JCAMP-DX are ASCII-files with file heads containing information about the data and their origin, etc., and they may contain both X-data (spectra) and Y-data (concentrations).

Only the most essential information of the JCAMP-DX file will be imported, and some of it gets into the Instrument Parameters field of the Unscrambler file. The first title in the .JDX-file will be used, and truncated if too long. If several .JDX-files are imported and saved on the same Unscrambler file, the comments field will contain the information from the last imported .JDX-file.

JCAMP "X-values" (usually wavelengths) become X-variable names, while JCAMP "Y-values" become X-variable values. "Concentrations" are interpreted as Y-variables. Variable names are truncated to 8 characters. The first 8 (non blank) characters of the “Sample description” are used as sample names. So when coding samples at scanning - choose codes with care and you will be able to use the coding information at interpretation of the calibration models. Unfortunately there are different dialects of JCAMP-DX, so in some cases you may lose eg. sample names if they were used erroneously in the original file.

Here we show examples of the XYDATA and XYPOINTS formats. The XYPOINT options demands more disk space than XYDATA.

JCAMP-DX XYPOINTS

The example below shows only one sample. Items in italic do not have to be included.

```
##TITLE= DMCAL.DAT to DMCAL19.DAT using FILTER1.DAT wavelengths
##JCAMP-DX= 4.24  $$IDAS  1.40
##DATA TYPE= LINK
##ORIGIN= Bran+Luebbe Analyzing Technologies
##OWNER= Applications Laboratory
##DATE= 92/ 6/10  $$ WED
##TIME=  1: 0: 3
##BLOCKS=  14
##TITLE=  DMCAL.DAT to DMCAL19.DAT using FILTER1.DAT wavelengths
##JCAMP-DX= 4.24  $IDAS  1.40
##DATA TYPE= NEAR INFRARED SPECTRUM
##ORIGIN= Bran+Luebbe Analyzing Technologies
##OWNER= Applications Laboratory
##SAMPLE DESCRIPTION= WHE202CH          $$      1.00
##SAMPLING PROCEDURE= DIFFUSE REFLECTION
##DATA PROCESSING= LOG(1/R)
##XUNITS= NANOMETERS
##YUNITS= ABSORBANCE
##XFACTOR= 1.0
##YFACTOR=  0.000001
##FIRSTX=   1445
##LASTX=   2348
##FIRSTY=  0.652170
##MINY=  0.552445
##MAXY=  1.258505
##NPOINTS=  19
##CONCENTRATIONS= (NCU)
(<CARBOHYDRATE>,   89.400, %)
(<PROTEIN     >,    9.410, %)
##XYPOINTS= (XY..XY)
  1445,  652170;   1680,  555209;   1722,  606660;   1734,  612745;
  1759,  604142;   1778,  575455;   1818,  552445;   1940,  631510;
  1982,  657704;   2100, 1188830;   2139, 1082772;   2180, 1008640;
  2190,  999405;   2208,  951049;   2230,  978299;   2270, 1198344;
  2310, 1258505;   2336, 1209149;   2348, 1153169;
##END=
```

JCAMP-DX XYDATA

The example below shows only one sample. Items in italic do not have to be included.

```
##TITLE= InfraAlyzer 500 (5 NM Intervals)
##JCAMP-DX= 4.24  $$IDAS  1.40
##DATA TYPE= LINK
##ORIGIN= Bran+Luebbe Analyzing Technologies
##OWNER= Applications Laboratory
##DATE= 92/ 7/ 9  $$ THU
##TIME= 20:53:17
##BLOCKS=  14
##TITLE= InfraAlyzer 500 (5 NM Intervals)
##JCAMP-DX= 4.24  $IDAS  1.40
##DATA TYPE= NEAR INFRARED SPECTRUM
##ORIGIN= Bran+Luebbe Analyzing Technologies
##OWNER= Applications Laboratory
##SAMPLE DESCRIPTION= COF12BUS      $$      1.00
##SAMPLING PROCEDURE= DIFFUSE REFLECTION
##DATA PROCESSING= LOG(1/R)
##XUNITS= NANOMETERS
##YUNITS= ABSORBANCE
##XFACTOR= 1.0
##YFACTOR=  0.000001
##FIRSTX=   1100
##LASTX=   2500
##FIRSTY=  0.139460
##MINY=  0.131600
##MAXY=  1.380070
##NPOINTS= 281
##CONCENTRATIONS= (NCU)
(<CARBOHYDRATE>,   89.400, %)
(<PROTEIN     >,    9.410, %)
##DELTAX=   5
##XYDATA= (X++(Y..Y))
  1100  139459  137435  135089  133060  131669  131599  133794  138899
  1140  145740  151897  158459  167527  180800  195522  206585  216499
  ...
  ...
  2460 1378929 1379632 1378464 1374972 1378929 1376837 1372945 1377632
  2500 1380069
##END=
```

Instrument Parameters for JCAMP Files

The appropriate parameters in this field will be written to the JCAMP exported file.

Please feel free to include more parameters in the file if necessary . You may type whatever you like into the field, but only text on the format ##KEYWORD = ….., as listed below, will be used during export.

JCAMP keywords

Keyword	Legal values
##AVERAGE=	INTEGER*4 > 0
##GAIN=	REAL*4 >= 0.0
##BASELINEC=	YES or NO.
##APCOM=	String60
##JCAMP-DX=	String
##ORIGIN=	String

NSAS File Format

We do not describe the NSAS file format here, but restrict the description to describing the instrument parameters imported from NSAS data files.

Instrument Parameters from NSAS Files

NSAS Data Import will read information in the NSAS data file which has no natural place in The Unscrambler's file format into the Instrument Info block under specific keywords. Similarly, NSAS/Vision Model Export will look for a relevant subset of these keywords and, if found, it will place the values in the corresponding places in the NSAS/Vision Model file.

The NSAS/Vision keywords are listed below.

NSAS/Vision keywords

Keyword	Legal values
NSAS_InstrumentModel	String representing integer > 0
NSAS_AmpType	String: *Reflectance* *Transmittance* *(Reflect/Reflect)* *(Reflect/Transmit)* *(Transmit/Reflect)* *(Transmit/Transmit)* *Not used*
NSAS_CellType	String: *Standard sample cup* *Manual* *Web analyzer* *Coarse sample* *Remote reflectance* *Powder module*

Keyword	Legal values
	High fat/moisture *Rotating drawer* *Flow-through liquid* *Cuvette* *Paste cell* *Cuvette cell* *3 mm liquid cell* *30 mm liquid cell* *Coarse sample with sample dump*
NSAS_Volume	String: *1/4 full* *1/2 full* *3/4 full* *Completely full*
NSAS_NumScans	String representing integer > 0
NSAS_HasSampleTransport	String: *Yes* *No*
NSAS_ReferenceAcquiredInRefPos	String: *Yes* *No*
NSAS_SampleAcquiredInSamPos	String: *Yes* *No*
NSAS_OnlineInstrument	String: *Yes* *No*
NSAS_Math1_Type NSAS_Math2_Type NSAS_Math3_Type	String representing integer > 0: 1 = “N-point smooth" 2 = “Reflective energy" 3 = “Kubelka-Munk" 4 = “1st derivative" 5 = “2nd derivative" 6 = “3rd derivative" 7 = “4th derivative" 8 = “Savitsky & Golay" 9 = “Divide by wavelength" 10 = “Fourier transform" 11 = “Correct for reference changes" 13 = “Full MSC" 21 = “N-point smooth" 22 = “1st derivative" 23 = “2nd derivative" 31 = “Savitzky-Golay first derivative"
NSAS_Math1_SegmentSize NSAS_Math2_SegmentSize NSAS_Math3_SegmentSize	String representing integer > 0
NSAS_Math1_GapSize	String representing integer > 0

Keyword	Legal values
NSAS_Math2_GapSize NSAS_Math3_GapSize	
NSAS_Math1_DivisorPoint NSAS_Math2_DivisorPoint NSAS_Math3_DivisorPoint	String representing integer > 0
NSAS_Math1_SubtractionPoint NSAS_Math2_SubtractionPoint NSAS_Math3_SubtractionPoint	String representing integer > 0
NSAS_NumberOfConstituents	String representing integer > 0
NSAS_NumberOfDataPoints	String representing integer > 0
NSAS_StartingWaveLength	String representing integer > 0
NSAS_EndWaveLength	String representing integer > 0
NSAS_CreationDay	String representing integer > 0
NSAS_CreationMonth	String representing integer > 0
NSAS_CreationYear	String representing integer > 0
NSAS_CreationHour	String representing integer > 0
NSAS_CreationMinute	String representing integer > 0
NSAS_CreationSecond	String representing integer > 0

GRAMS File Format

This format is used by software packages from Galactic (Grams32, PLSPlus/IQ), and is also used by many spectroscopy instrument vendors.

The data are stored in two different file types. Spectra are stored in binary files with the .SPC extension, and constituents are stored in ASCII files with the .CFL extension. The two file types are connected so that if a .CFL file is imported into The Unscrambler, both spectra and constituents are read. If a .SPC file is imported, only the spectra are read.

SPC "X-values" (usually wavelengths) are imported as X-variable names. SPC "Y-values" are imported as X-variable values. Constituents in .CFL files are imported as Y-variables.

Some SPC files contain a log block. The binary part of the log block (which usually contains the imaginary part of complex spectral data) is not imported. The ASCII part of the log is stored in the "Instrument Parameters" field of the new Unscrambler file.

Result Matrices

The table below contains the names and contents of all matrices used in The Unscrambler.

Result matrices

Matrix Name	Description	Matrix Name	Description	Matrix Name	Description
AnovaSum	Analysis of variance error	JackSignifP	p-values for X-loadings	ResYValSamp	Residual Sample Y-variance, validation
AnovaTab	Analysis of variance table	JackSignifQ	p-values for Y-loadings	ResYValTot	Total residual Y-variance, validation
AnovaTab0	Analysis of variance row B0	JackSignifW	p-values for loadings weights	ResYValTotCVS	Total residual Y-variance for each CV-segment
B	B-coefficients *)	JackVarBw	Variance for weighted regression coefficients	ResYValVar	Y-variables residual variance, validation
B0	B0-coefficients *)	JackVarPax	Variance for X-loadings	RMSECal	RMS error of calibration
B0W	Weighted B0-coefficients	JackVarQay	Variance for Y-loadings	RMSEVal	RMS error of validation
Bdiagnosis	B-diagnosis	JackVarTai	Variance for scores	S0	S0
BdiagnosisW	Bw-diagnosis	JackVarWax	Variance for loading weights	Si	Si
bSTDError	Standard error in b-coeff	LackOfFit	Lack of fit	Si/S0	Ratios of Si/S0
bSTDError0	Standard error in B0	LevelledVars	All design and cat. vars in data	SigLevelEff	Effect levels for significance
BW	Weighted B-coefficients	Members	Classification table	SquSum	Square sums
CanonDir	Canonical directions	MMSCode	Max/Min/Saddle, pt type	StatCorr	Statistics, correlations
CentI&S	Centers for Int. and Squares	MMSPtCoord	Max/Min/Saddle, variable values	StatMean	Statistics, mean values
DiscrPower	Discrimination power	MMSyPred	Max/Min/Saddle, Y-predicted	StatNSamp	Statistics, number of samples
Effects	Estimated effects	ModCheck	Model check	StatPercentile	Statistics, percentiles
EigenValues	Eigenvalues	ModelDist(*m*)	Model distance, from model m	StatPrecision	Statistics, Sdev of measurements
Eix	X-residuals	ModelDistance	Model distance	StatSDev	Statistics, standard deviation

*) Analysis of Effects uses coded levels of the X-variables for calculating the B-coefficients. In all other analyses, B-coefficients are based on real values of the X-variables. However, in Response Surface analysis, the data are centered before the calculations, and this affects the value of B0.

Result matrices (continued)

Matrix Name	Description	Matrix Name	Description	Matrix Name	Description
ExpXCalSamp	Explained X-variance per sample, calibration	ModelPower	Model power, classification	StudentRes	Studentized residuals
ExpXCalTot	Total explained X-variance, calibration	MultCompMean	Multiple comp. Mean	Tai	Scores
ExpXCalVar	Explained X-variance per variable, calibration	MultCompNSamp	Multiple comp. no. of samples	TaiClass(*m*)	Scores in classification
ExpXValSamp	Explained X-variance per sample, validation	MultCompT	Multiple comp. acceptance lvl	tFpValues	t/F/p-values
ExpXValTot	Total explained X-variance, validation	MultCorrCal	Multiple correlation, calibration	TFpValues0	t/F/p-values B0
ExpXValTotCVS	Total explained X-variance for each CV-segment	MultCorrVal	Multiple correlation, validation	TValCurv	t-values for curvature test
ExpXValVar	Explained X-variance per variable, validation	Pax	X-loadings	Uai	Preliminary scores
ExpYCalSamp	Explained Y-variance per sample, calibration	PCXVarCorr	(Xraw, scores) correlation	WaxPriX	Loading weights, primary X
ExpYCalTot	Total explained Y-variance, calibration	PCYVarCorr	(Yraw, scores) correlation	WaxSecX	Loading weights, secondary X
ExpYCalVar	Explained Y-variance per variable, calibration	PercentileMeas	Percentiles, measured	Wax	Loading weights
ExpYValSamp	Explained Y-variance per sample, validation	PercentilePred	Percentiles, predicted	WeightI&S	Weights for Int. and Squares
ExpYValTot	Total explained Y-variance, validation	pValCurv	p-values for curvature test	xBias	X-bias
ExpYValTotCVS	Total explained Y-variance for each CV-segment	pValEff	p-values for different effects	xCent	Model center X
ExpYValVar	Explained Y-variance per variable, validation	Qay	Y-loadings	xRaw	X raw data set
ExtraVal	Extra validation	ResPriXCalVar	Primary X-residual variance, calibration	Xweight	X-weights
ExtraValPred	Extra validation	ResPriXValVar	Primary X-residual variance, validation	xWeighted	Weighted raw X-variables
Fiy	Y-residuals	ResSecXCalVar	Secondary X-residual variance, calibration	yBias	Y-Bias

Result matrices (continued)

Matrix Name	Description	Matrix Name	Description	Matrix Name	Description
FValEff	F and Ψ-values for effects	ResSecXValVar	Secondary X-residual variance, validation	yCent	Model center Y
Hi	Sample leverage, cal/val	ResXCalSamp	Residual Sample X-variance, calibration	yDeviation	Y-deviation
HiCalMean	Mean leverage of calibr. Samples	ResXCalTot	Total residual X-variance, calibration	yPredCal	Y-predicted, cal
HiClass(*m*)	Sample leverage	ResXCalVar	X-variables residual variance, calibration	yPredicted	Y-predicted
Hx	X-var leverage	ResXValSamp	Residual Sample X-variance, validation	yPredSTDErr	Standard error in y-predicted
JackBw	B-coefficients for each CV segment	ResXValTot	Total residual X-variance, validation	yPredVal	Y-Predicted, val
JackRotPax	Rotated X-loadings for each CV segment	ResXValTotCVS	Total residual X-variance for each CV-segment	yRaw	Y raw data set
JackRotQay	Rotated Y-loadings for each CV segment	ResXValVar	X-variables residual variance, validation	yReference	Y-data for reference samples
JackRotTai	Rotated scores for each CV segment	ResYCalSamp	Residual Sample Y-variance, calibration	yRefPred	Reference Y-variables
JackRotWax	Rotated loading weights for each CV segment	ResYCalTot	Total residual Y-variance, calibration	yWeight	Y-weights
JackSignifB	p-values for regression coefficients	ResYCalVar	Y-variables residual variance, calibration	yWeighted	Weighted raw Y-variables

Abbreviations Used in Log Files

The log file keeps track of every change that is performed on the data table. The log is opened from File - Properties. The table below lists and describes all abbreviations used in The Unscrambler.

Abbreviations

Main	Description	Secondary	Description
ANA	Analysis…	AOE	Analysis of Effects
		CLA	Classification
		MLR	Multiple Linear Regression
		PCA	Principal Component Analysis

Abbreviations (continued)

Main	Description	Secondary	Description
		PCR	Principal Component Regression
		PL1	Partial Least Squares 1
		PL2	Partial Least Squares 2
		PRE	Prediction
		RES	Response Surface Analysis
		STA	Statistics
APP	Append…	SAM	Sample
		VAR	Variable
COM	Compute…	MAT	Matrix
		VEC	Vector
DEL	Delete…	SAM	Sample
		VAR	Variable
IMP	Import	-	
INS	Insert…	SAM	Sample
		VAR	Variable
REP	Replace	-	
SHI	Shift Variables	-	
SOR	Sort Samples	-	
TRA	Transform…	ATR	Absorbance to Reflectance
		BAS	Baseline
		DNO	Norris Derivative
		DSG	S. Golay Derivative
		MNO	Maximum Normalization
		MSC	Multiplicative Scatter Correction
		NOI	Added Noise
		NOR	Mean Normalization
		RED	Reduce
		RNO	Range Normalization
		RTA	Reflectance to Absorbance
		RTK	Reflectance to Kubelka-Munck
		SMA	Moving Average Smoothing
		SSG	S. Golay Smoothing
		TSP	Transpose
		USR	User-Defined

Compatibility With Other Software And Instruments

The table below contains an incomplete list of how you can connect The Unscrambler to other software packages and instruments.

Compatibility with other software and instruments

Supplier	Software	Instr.	Writes Unsc-format or includes converter	Unsc 6.0 import	Reads Unsc model files
Analect	FX 70 (DOS) FX 80 (Win) D/A only, no calibration or prediction	Diamond 20 (lab)	Converts to flat ASCII, 1 file pr spectrum [7]	ASCII	
ASD			ASCII		
BioRad		FTIR	No	Import JCAMP-DX	
Bomem	D/A: Win BomemEasy or Bomem Grams Prediction: Airs for QA/QC, or CAAP			Import JCAMP-DX or ASCII.	U6: Soon
Bran+Luebbe	Sesame	Infralyzer	U5, Unsc Ascii	Import U5	Yes, U5
Bran+Luebbe	IDAS	Infralyzer	No	Import JCAMP-DX	No - manual[5]
Bran+Luebbe	APC	Infralyzer	No		No - manual[5]
Bran+Luebbe	ICAP	Infraprover	No	Import ASCII	No
Brimrose	Luminar 2000 (process) Lumunar for lab		DOS: conversion to U5 or flat ASCII. Win: flat ASCII or Lotus wk?.	Import U5 Import flat ASCII or Lotus .wk?	Soon
Bruker	OPUS	IFS28XN Vector 22/N	ASCII or JCAMP DX Single spectrum files	Probably import ASCII or JCAMP	
Foss	Tracker	Meatspec		Import Tracker	Yes. Tracker exp.
Foss	Tracker	Grainspec		Import Tracker	Yes. Tracker exp.
Foss		Milkoscan FT 120 incl selection of wavelengths	ASCII	Import ASCII	
Galactic	Grams			Import JCAMP-DX Import Grams: Soon	Soon
Galactic	Spectracalc			Import JCAMP-DX	
LT	LT1200+ (NIR)	Spectrametrix v 1.91 and earlier (D/A, instr. Control, prediction) LightCal (PCR, PLS)	BIN2ASC conversion to ASCII (1 file pr spectrum) with header. Must be edited and Lotus wks (remove first line). Converter available.	Import Flat ASCII. Import Lotus wk3. Import Unsc ASCII.	
Nicolet				Import JCAMP-	

Supplier	Software	Instr.	Writes Unsc-format or includes converter	Unsc 6.0 import	Reads Unsc model files
				DX	
Nicolet	OMNIC			Import JCAMP-DX or CVS ASCII	
NIRSystems	NSAS	NIRS	No	Import JCAMP or NSAS	Yes
NIRSystems	ISI	NIRS	No	Import JCAMP or NSAS	No
Perkin-Elmer	PioNIR		U5	Import U5	Soon
Perkin-Elmer	Several e.g. Grams	e.g. P-E 2000 FTIR	No	Import JCAMP-DX	No
Perkin-Elmer	Quant			Import JCAMP-DX	
Perten	Inframatic	NIR program package		Flat ASCII	No - manual
Polytech/IKS		X-dap diode aray	U5	Import U5	Yes, U5 export
Tecator	Inframaker	Infratech	U5, Unsc ASCII	Import U5	Yes, U5 export
UOP GW	GW Method maker	GW series	U5	Import U5	Yes, U5 export
Varian	Cary		No	Import JCAMP-DX	
Yokogawa		Infraspec		Import JCAMP-DX	U6: Yes
ASCII, used by most software packages	ASCII flat, Unsc-ASCII			Import ASCII	Export any data and results as ASCII data
JCAMP-DX, used by many instruments	JCAMP-DX			Import JCAMP-DX Merge imports many files	U5: Export macro JCAMP-DX u6: Export data or B-vector as JCAMP-DX
EXCEL	Excel		No	Import .xls (v5 workbook, v4 worksheet)	
LOTUS 1-2-3	Lotus		No	Import .wi3 .wk4	

Note: The Unscrambler allows you to merge several JCAMP-DX files upon Import.

Building User-Defined Transformations

This chapter describes the interface between The Unscrambler and the optionally installed User-Defined Transformation (UDT) or User-Defined Analysis (UDA) components.

These components are available in the Editor from resp. **Modify - Transform - User-defined** and **Modify - Transform - User-defined Analysis** (the latter only with 3-D data tables), and can apply a variety of transformations to a data set or run a new type of analysis on 3-D data. The installed component takes over for a while and complements the standard features of The Unscrambler. Such components might be developed by the user or any third party vendor, and installed separately on the same computer that The Unscrambler is run on.

The text below assumes a minimal required knowledge of what the Component Object Model (COM) is, and how to implement COM interfaces & objects. Alternatively, a basic knowledge of building OLE Automation (IDispatch) classes, or a development environment that supports/simplifies its use (such as Microsoft Visual Basic and Matlab), will be sufficient.

In what follows, we will refer mostly to UDT's, which were available first, but the same principles apply to the creation of UDA's.

Installation of UDT Components

Preferably, a UDT components installation program should automatically update the Registry with all entries required, including the category ID for UDT classes. When there is no installation utility, or the file has been copied manually, the DLL can be registered using the Windows REGSVR32 utility as follows:

```
regsvr32 MyFile.dll
```

This requires that the DLL is self-registering (i.e. implements the *DllRegisterServer* and *DllUnregisterServer* functions). The DLL can be removed from the registry ("uninstalled") by typing:

```
regsvr32 /u MyFile.dll
```

In some cases, this may register the available COM classes correctly, but not identifying it as "implementing the UDT category". In such cases the class can be located and registered from within The Unscrambler's UDT dialog.

Another way to do this is by means of Registry import files (.reg). A template file is provided in the Unscrambler DATA\UDT directory. This file must be copied/edited with your own class ID, and then imported either using the Registry editor, or by double-clicking the .reg file from an Explorer window.

Requirements for being Recognized as UDT Component

For a User-Defined Transformation component to be recognized as such by The Unscrambler, and listed in the dialog box for applying a UDT, it must satisfy certain requirements:

- The class must be registered (standard entries in the Registry).
- The class must be "in-process" (server or handle) – usually in a .DLL file. This requirement is due to memory addresses being passed between The Unscrambler and the UDT component, so they must reside in the same address space.
- The component must implement either
  a) the IUnscTransformation interface (COM), or
  b) the IDispatch interface (OLE Automation) providing all method names found in the IUnscTransformation interface.
- The class must be marked (in the Registry) as implementing the category CATID_UnscUserDefTrans. This can, however, be done from inside The Unscrambler by manually selecting the component from a list.

UUIDs Defined

Below are the official UUIDs – universally unique identifiers (also called GUIDs – globally unique identifiers), that are defined by The Unscrambler's User-Defined Transformation interface. These will be needed especially when using the COM interface.

Official interface & category IDs

Name/symbolic ID	GUID/UUID (registry format)
CATID_UnscUserDefTrans	{E809A521-BDA7-11D2-96A4-006008E7ED62}
IID_IUnscDataArray	{26823FC1-B781-11D2-96A4-006008E7ED62}
IID_IUnscTransformation	{26823FC2-B781-11D2-96A4-006008E7ED62}
LIBID_UDTransLib	{08D48441-C12D-11D2-96A4-006008E7ED62}

In addition to these, class IDs will need to be defined by each UDT component itself.

Error Codes & Return Status

All methods defined by the interfaces *IUnscTransformation* and *IUnscDataArray* return standard COM error status codes (type HRESULT). The corresponding OLE automation interfaces also return the same set of error codes, but one should keep in mind that a wider range of these error codes are likely to occur in this case – such as a run-time type mismatch (DISP_E_TYPEMISMATCH) or parameter count mismatch (DISP_E_BADPARAMCOUNT).

The general rule is that a positive return value (or bit 31=0) is considered non-fatal, while a negative value (or bit 31=1) should be treated as a fatal error. In practice when using the *IUnsc** interfaces, the only "successful" return status in use is S_OK (=0), and everything else is an error. The code that uses these methods should be prepared to handle any return status code.

COM Interfaces

Except for the IUnknown methods **AddRef** and **Release**, all methods return a standard COM result code of type HRESULT (a 32-bit signed integer), where S_OK (=0) indicates a successful method invocation.

IUnscTransformation

The **IUnscTransformation** interface enables an UDT component to respond to queries from The Unscrambler, including saving/restoring its settings and performing its transformation to a data array. The interface is implemented by the UDT component, and is used by The Unscrambler.

The methods should normally not require any user intervention (e.g., dialog boxes), with the exception of the **ModifyParameters** method which is excepted to do just that. Another situation where it might be handy to use dialog boxes, might be for debugging purposes during the development of a UDT component. Dialog boxes should be modal, and destroyed before returning from the method.

Methods in VTable Order:

IUnknown methods

- **QueryInterface** Returns pointers to supported interfaces
- **AddRef** Increments the reference count
- **Release** Decrements the reference count

IUnscTransformation methods

- **GetCapabilities** Gets capability flags (bit map)
- **ModifyParameters** Shows dialog for modifying parameters
- **GetParamString** Gets string describing current parameters
- **GetStorageSize** Gets required size of storage buffer
- **Store** Saves instance-specific data in a storage buffer
- **Restore** Restores instance-specific data from a storage buffer
- **ApplyOn** Applies the transformation on a data array

IUnscTransformation::GetCapabilities

HRESULT GetCapabilities

(long nNumColumns, long nNumRows, long *pnCapabilityMap)

Gets capability map (bit flags), describing the UDT components cabilities, possibly depending on the current size of the scope of operation (number of columns and rows). Here the component may indicate its desire to delete or insert columns/rows, and the availability of a suitable parameter configuration dialog that the user can access.

The following bits are used in the returned 32-bit map:

Bits 31 … 6	5	4	3	2	1	0
Reserved	*Rd*	*Ri*	*Cd*	*Ci*	*res.*	*P*

where

- *Rd* = "Row Delete." Set to 1 if the UDT component might want to delete one or more rows from a data array of the size given.
- *Ri* = "Row Insert." Set to 1 if the UDT component might want to insert one or more new rows in the data array.
- *Cd* = "Column Delete." Set to 1 if the UDT component might want to delete one or more columns from the data array.
- *Ci* = "Column Insert." Set to 1 if the UDT component might want to insert one or more new columns in the data array.

All other bits are reserved for future extensions and should be set to 0.

The Unscrambler (through the *IUnscDataArray* interface) refuses some operations (insert, delete) if the corresponding action is not indicated in the capability map.

Parameters

- nNumColumns [in] Number of columns in current scope
- nNumRows [in] Number of rows in current scope
- pnCapabilityMap [out] Bit map with capability map (assuming current scope)

IUnscTransformation::ModifyParameters

HRESULT ModifyParameters

(long nNumColumns, long nNumRows)

Modifies transfomation-specific parameters. During this call, the UDT component is supposed to present to the user a suitable (modal) dialog box where various parameters or transformation settings may be inspected and changed.

Parameters

- nNumColumns [in] Number of columns in current scope
- nNumRows [in] Number of rows in current scope

IUnscTransformation::GetParamString

HRESULT GetParamString

(BSTR *psParams)

Gets a string describing the current parameters. This string is displayed in the Transformation Log. The string can be anything the component finds useful, or empty if nothing needs to be displayed.

Parameter

- psParams [out] Parameter description string

IUnscTransformation::GetStorageSize

HRESULT GetStorageSize

(long *pnBufferSize)

Gets required size (in bytes) for storing instance-specific data. This method is called every time The Unscrambler needs to save the transformation parameters to a file, and is

used to allocate a buffer where the UDT component can put its data (when **Store** is called).

Parameter

- pnBufferSize [out] Required size of buffer

IUnscTransformation::Store

HRESULT Store

(long nBufferSize, long pBuffer)

Stores all instance-specific data to specified buffer. All data including user settings and parameters necessary to restore the UDT object to its current state at a later time (using **Restore**) should be saved.

Parameters

- nBufferSize [in] Size allocated in buffer (number of bytes)
- pBuffer [out] Address of (pointer to) buffer to store the instance-specific data block

Note: The buffer address is transferred as a 32-bit signed integer value. This should be interpreted as the starting memory address of the buffer.

IUnscTransformation::Restore

HRESULT Restore

(long nBufferSize, long pBuffer)

Restores the object from instance-specific data. This should reset the object to the same state as it had when the given buffer once was written to (via **Save**), regardless of the object's current state.

Parameters

- nBufferSize [in] Size of buffer data (in bytes)
- pBuffer [in] Address of (pointer to) buffer where the instance-specific data block is stored

Note: The buffer address is transferred as a 32-bit signed integer value. This should be interpreted as the starting memory address of the buffer.

IUnscTransformation::ApplyOn

HRESULT ApplyOn

(IUnscDataArray **ppDataArray)

Applies the (user-defined) transformation on a data array. The transformation should be consistent with those potensial changes reported by **GetCapabilities**, as well as those set by the last call to **ModifyParameters** or the UDT components default settings. A double-indirect pointer to an IUnscDataArray interface is passed from The Unscrambler, and provides the means by which the component may get its input data, as well as store its result. Upon returning from this method, the UDT components should release all additional references to the IUnscDataArray object given; i.e. it must not keep any references to it beyond the scope of the call to **ApplyOn** itself.

Parameter

- ppDataArray [in, out] Pointer to interface pointer of data array to transform

IUnscDataArray

The **IUnscDataArray** interface enables an UDT component to access & update data within The Unscrambler, which is the data that it operates on during the call to **IUnscTransformation::ApplyOn**. The interface is not normally implemented by the UDT component, but is provided (and implemented) by The Unscrambler. The UDT component is only allowed to perform those changes to the data set that this interface supports.

The data array is a virtual 2-dimensional matrix of single-precision (32-bit) floating point values, representing the selected scope of operation. The rows of this matrix are numbered 0 thru (*NumRows*–1), while the columns are numbered 0 thru (*NumColumns*–1). The selected scope may contain a subset of the full matrix in The Unscrambler. For instance, only some of the variables might be selected or included in the operation, corresponding to the virtual columns in the provided *IUnscDataArray* interface. The principle is the same for the selected samples, which become virtual rows in the *IUnscDataArray* interface. Any interaction & square variables contained in a selected variable set are *not* counted nor included in the virtual data array.

Example:

The user selects the variables 2–4, 9, 14, 32 and 33, as well as all 25 samples as the scope for a user-defined transformation. This adds up to a total of 7 columns (= the selected variables), numbered 0 thru 6, and 25 rows in the virtual matrix, the *IUnscDataArray* interface, provided when the transformation is applied.

Methods in VTable Order:

IUnknown methods

- **QueryInterface** Returns pointers to supported interfaces
- **AddRef** Increments the reference count
- **Release** Decrements the reference count

IUnscDataArray methods

- **GetDimensions** Gets size/dimensions of the data array
- **GetMissing** Gets the value representing 'missing'
- **GetValue** Gets the value of a single element
- **SetValue** Sets the value of a single element
- **GetColumnVector** Gets vector of values from a column
- **SetColumnVector** Updates vector of values in a column
- **GetRowVector** Gets vector of values from a row
- **SetRowVector** Updates vector of values in a row
- **InsertColumns** Inserts/appends new columns into the data array
- **InsertRows** Inserts/appends new rows into the data array
- **DeleteColumns** Deletes one or more columns in the data array
- **DeleteRows** Deletes one or more rows in the data array

IUnscDataArray::GetDimensions

HRESULT GetDimensions

(long *pnNumColumns, long *pnNumRows)

Gets size (dimensions) of the data array. Row & columns numbering of the virtual data array always start at 0 (zero) for the first vector.

The size is valid until an insert or delete operation is done, after which the UDT component must either keep track of the changes itself, or call this method again to get updated dimensions.

Parameters

- pnNumColumns [out] Number of columns in data array
- pnNumRows [out] Number of rows in data array

IUnscDataArray::GetMissing

HRESULT GetMissing(float *pfMissingValue)

Gets the value (binary representation) representing ‘missing’ data in the data array. This value can be used for setting elements to ‘missing’, or for testing elements for the presence of ‘missing’ value (absence of data).

Parameter

- pfMissingValue [out] Value representing 'missing' data

IUnscDataArray::GetValue

HRESULT GetValue

(long nColumn, long nRow, float *pfValue)

Gets the current value of 1 element in the data array.

Parameters

- nColumn [in] Selected column
- nRow [in] Selected row
- pfValue [out] Value of that cell

IUnscDataArray::SetValue

HRESULT SetValue

(long nColumn, long nRow, float fValue)

Sets or updates the value of 1 element in the data array.

Parameters

- nColumn [in] Selected column
- nRow [in] Selected row
- fValue [in] New value of that cell

IUnscDataArray::GetColumnVector

HRESULT GetColumnVector

(long nColumn, long nStartRow, long nMaxNumRows, float *afVector, long *pnNumRows)

Gets vector of values from a column of the data array. The caller must allocate the array of values to store the returned vector. Fewer than the requested number of rows are returned if that would exceed the last row of the data array. E.g. if the data array contains 10 rows, and 5 elements are requested starting at row #7, then only 3 elements will be returned (#7, #8, and #9). An S_OK return status is reported even if there are fewer values returned than requested, so this must be checked separately.

Parameters

- nColumn [in] Selected column
- nStartRow [in] 1st selected row
- nMaxNumRows [in] Max. number of rows (values) to return
- afVector [out] Array of values from requested column
- pnNumRows [out] Number of values (rows) actually returned

IUnscDataArray::SetColumnVector

HRESULT SetColumnVector

(long nColumn, long nStartRow, long nNumRows, float *afVector)

Updates a vector of values in a column of the data array. The vector may start at any legal row index, but must not exceed the column vector's last row.

Parameters

- nColumn [in] Selected column
- nStartRow [in] 1st selected row
- nNumRows [in] Number of rows (values) to update
- afVector [in] Array of (new) vector values

IUnscDataArray::GetRowVector

HRESULT GetRowVector

(long nStartColumn, long nRow, long nMaxNumColumns, float *afVector, long *pnNumColumns)

Gets vector of values from a row of the data array. The caller must allocate the array of values to store the returned vector. Fewer than the requested number of columns are returned if that would exceed the last column of the data array. E.g. if the data array contains 10 columns, and 5 elements are requested starting at column #7, then only 3 elements will be returned (#7, #8, and #9). An S_OK return status is reported even if there are fewer values returned than requested, so this must be checked separately.

Parameters

- nStartColumn [in] 1st selected column
- nRow [in] Selected row
- nMaxNumColumns [in] Max. number of columns (values) to return
- afVector [out] Array of values from requested row
- pnNumColumns [out] Number of values (columns) actually returned

IUnscDataArray::SetRowVector

HRESULT SetRowVector

(long nStartColumn, long nRow, long nNumColumns, float *afVector)

Updates a vector of values in a row of the data array. The vector may start at any legal column index, but must not exceed the row vector's last column.

Parameters

- nStartColumn [in] 1st selected column
- nRow [in] Selected row
- nNumColumns [in] Number of columns (values) to update
- afVector [in] Array of (new) vector values

IUnscDataArray::InsertColumns

HRESULT InsertColumns

(long nAfterColumn, long nNumNewColumns, BSTR *psName)

Inserts (or appends) one or more new columns in the data array. The new columns will be inserted just to the right of the column specified by *nAfterColumn*. Columns can be inserted before the 1st column by specifying *nAfterColumn* = –1.

All new columns are given the name specified by *psName*, or The Unscrambler's default column name if the *psName* string is empty. The values of all elements in the new columns are initially set to 'missing'.

Parameters

- nAfterColumn [in] Index of the column, after which the new columns will be inserted.

- nNumNewColumns [in] Number of new columns to insert
- psName [in] Name of the new columns (heading)

IUnscDataArray::InsertRows

HRESULT InsertRows

(long nAfterRow, long nNumNewRows, BSTR *psName)

Inserts (or appends) one or more new rows in the data array. The new rows will be inserted just below the row specified by *nAfterRow*. Rows can be inserted before the 1st row by specifying *nAfterRow* = –1.

All new rows are given the name specified by *psName*, or The Unscrambler's default row name if the *psName* string is empty. The values of all elements in the new rows are initially set to 'missing'.

Parameters

- nAfterRow [in] Index of the row, after which the new rows will be inserted.
- nNumNewRows [in] Number of new rows to insert
- psName [in] Name of the new rows (heading)

IUnscDataArray::DeleteColumns

HRESULT DeleteColumns(long nFirstColumn, long nLastColumn)

Deletes one or more columns in the data array starting at the column specified by *nFirstColumn*. The last column deleted is the one specified by *nLastColumn*. To delete a single column, set *nFirstColumn* = *nLastColumn*.

Parameters

- nFirstColumn [in] Index of first column to remove
- nLastColumn [in] Index of last column to remove

IUnscDataArray::DeleteRows

HRESULT DeleteRows(long nFirstRow, long nLastRow)

Deletes one or more rows in the data array starting at the row specified by *nFirstRow*. The last row deleted is the one specified by *nLastRow*. To delete a single row, set *nFirstRow* = *nLastRow*.

Parameters

- nFirstRow [in] Index of first row to remove
- nLastRow[in] Index of last row to remove

OLE Automation Interfaces (IDispatch)

OLE Automation is a relatively thin functional layer on top of the basic COM layer. For the purposes of interfacing to The Unscrambler, one only needs to relate to the special COM interface *IDispatch*, which implements the basics of OLE Automation. Some development environments provide automated generation of code for using OLE Automation or other COM interfaces, maybe using template collections (e.g. Microsoft's Active Template Library, or ATL). Other languages (like Visual Basic) even hide the details of using the *IDispatch* interface, and let the user focus on the actual methods & properties provided instead.

For using OLE Automation with UDT components, one has to relate to two different automation objects, implementing the functionality similar to the corresponding two COM interfaces *IUnscTransformation* and *IUnscDataArray*.

For both interfaces, all method names are the same as those used in equivalent COM interface. The functionality of the *IUnscDataArray* is still implemented by The Unscrambler, but when passing the interface to the method corresponding to *IUnscTransformation::ApplyOn*, an *IDispatch* interface to an OLE Automation *DataArray* object is passed instead. The UDT has to implement the methods of the *IUnscTransformation* interface through its *IDispatch* interface.

All methods have the same parameters as in their corresponding COM interfaces, *except* they all get an extra parameter, **long *hRes**, added at the end. This parameter is a pointer to a 32-bit signed integer, and is used to return a duplicate of the HRESULT return status value. For some programming languages (e.g. Visual Basic) this facilitates checking/returning the return status, as it is not otherwise directly available. For languages that have easy access to the method's HRESULT return status, it is fully acceptable to send a NULL pointer for this trailing parameter whenever one of the *IUnscDataArray* methods is invoked in The Unscrambler. The Unscrambler, on the other hand, will always send a non-NULL pointer when invoking the *IUnscTransformation* methods, and will inspect the returned value of that parameter only when the ordinary HRESULT return status reports success (S_OK). In this case, the value of the **hRes** parameter is considered the method's "real" return status.

Development Files and Samples

All files necessary for developing User-Defined Transformation components are found the the sub-directory **UDT** of the target installation path for Unscrambler DATA. Here are the master Interface Desciption Language file (udtrans.idl) describing both the

IUnscTransformation and the *IUnscDataArray* interfaces, as well as its binary equivalent type library (udtrans.tlb) and automatically generated C/C++ header & implementation files.

Sample applications are found in the sub-directory **UDT\Samples**, and include samples in C++ using both MFC and ATL (w/ project file for Microsoft Visual Studio 5.0 or newer), as well as in Visual Basic (version 4.0 or newer required).

All sample transformations have their class name beginning with the letter 'z', which will make them appear towards the end of the (alphabetically sorted) list of available functions in the User-Defined Transformation dialog.

Bibliographical References

- McDonald, Wilks, JCAMP-DX: A Standard form for exchange of infrared spectra in computer readable form, Applied Spectroscopy, 1988, **42**(1)

Index