Manufacturing Management Software

Reference

Making IT Work

Save Time

Save Money

Improve Performance

Comprehenisve - Proven - Affordable

www.match-it.com

Smart
Funding Innovation

Table of Contents

Reference

This section contains various reference topics.

1 Import Filters

This topic describes how to set-up *import filters* and *import packages* to import information from CSV files (Comma Separated Value), or text files, into Match-IT. An *import filter* is a simple script that you write to describe how to translate a particular CSV file, or text file, format into an equivalent import package that is acceptable to Match-IT. An *import package* is a simple description of a related set of import records that you wish to import into Match-IT. The CSV format is a very simple and popular format for transferring information from one system to another. Most modern software packages can produce CSV files. The Match-IT importing system can also extract information from reports created from other systems that have been sent to a file. This note uses the *Sage Line 50* stock file and its stock explosion report as examples.

Note: These facilities are only useful when the import transformations are relatively simple. For more complex situations it's far better to write a Lua script.

1.1 Terminology

BNF:	Backus Naur Form. This is a notation for describing a grammar. Thing ::= A B (C **|** D) { E } [F] For example, the above means a 'Thing' is composed of an 'A' followed by a 'B', followed by a 'C' or a 'D', followed by any number of 'E's, followed by an optional 'F'.
CSV Field:	A single field from a CSV record or a text record. It can be one or more words, e.g. 'Stansted Road', or a number, e.g. 3.1459. See also *Fixed Field.*
CSV Record:	A single line from a CSV file. It consists of a series of CSV fields separated by a designated character (typically a comma). See also *Text Record.* Each line has the same format.
CSV Separator:	The character that is used to define the boundaries between fields in a CSV record.
Donor CSV File:	A CSV file consists of a series of CSV records, one per line. A 'donor' CSV file is one that is providing information from some other system for importing into Match-IT. See also *Text File.*
Element Separator:	Match-IT fields may consist of multiple elements. The element separator is the character that defines the boundaries between these elements. It is always a comma.
Field Element:	Match-IT fields may consist of multiple elements. For example, a customer contact is a single field in Match-IT, but it consists of two elements: the customer name and the contact name.
Field List:	A field list is required for each file presented to the import system. It consists of a single line of text; usually the first line in a CSV File. It defines the file ID the CSV records are intended for, and lists the field names present in each CSV record.
Filter:	See What is an import filter?
Fixed Field:	A single field from a *Text Record* that consists of a fixed number of characters in a fixed position in the record. See also *CSV Field.*
Group:	See What is a group?
Import:	This term is used to refer to the process of transferring information from

another system into the Match-IT system.

Ltype:	All fields in Match-IT files have an *LType* associated with them. An *LType* defines the type of information the field contains and how it is represented for importing purposes.
Match-IT CSV File:	A Match-IT CSV file is a CSV file that is acceptable to the import system. This usually requires a translation (by a filter) of a donor CSV file or a text file. Line 1 of the file must start with the file ID and be followed by a list of the names of the fields being imported. The order of the names in line 1 defines the order in the rest of the file. The order need not be the same as the actual Match-IT database. Unrecognised names will cause an error, and the import will abort, unless the name is preceded by a '!'. In this case the associated column in the CSV file will be ignored. Fields that exist in the Match-IT database but not defined in the field list will be filled with their default value.
Match-IT Field:	A Match-IT field is an item of information in one of its files. These fields are given values from the Match-IT CSV files presented to the import system.
Match-IT File:	The Match-IT files are the databases that define your Match-IT system. They are identified by a three letter ID for importing purposes.
Package Separator:	The character that is used to define the boundaries between fields in Match-IT CSV files. This is usually the vertical bar character |.
Package:	See What is an import package?
Quote Character:	Donor CSV files often put quote characters around non-numeric fields. E.g. "Stansted Road". They are significant in that they must be removed before the fields are imported. A default defines what Match-IT considers to be quote characters (usually " or ' or `).
Segment:	This term is used to refer to a part of a CSV or text file. The part is defined by a starting pattern and an ending pattern. The import system can be directed to ignore all lines up to, and including, the line containing the start pattern. It can also ignore all lines from that containing the end pattern to the end of the file, including the one containing the end pattern.
Syntax:	This term is used to refer to the allowed forms, or grammar, of a structured string. In this case, the import filter script.
Text File:	A collection of *Text Records*, one to a line. A text file is a source of information for importing. A *Text File* is similar to a Donor CSV File except that it may contain many different line formats.
Text Record:	A single line from a *Text File*. It consists of a series of fields (either CSV Fields or Fixed Fields). The fields present, and their format, may vary from line to line.

1.2 What is an Import Filter?

An *import filter* is a simple description of how to translate a CSV file, or a text file, of a particular format into an equivalent *import package*. The import filter is given a *Donor CSV File* or a *Text File*. The text of the filter describes how to interpret each field of each record of the file. The filter operates on one record at a time. The record is broken up into its fields and then re-assembled into one or more equivalent Match-IT records. Each input record may cause several Match-IT records, spanning several Match-IT files, to be created.

A limited field processing capability is available. This allows the translation of input fields into Match-IT fields to be dependent on the value of the fields. This conditional processing is available at the field level and the

Match-IT record level. It is illustrated in the examples given later.

In summary: a filter *translates* a file in one format into another format.

1.3 What is an Import Package?

An *import package* is a description of a related set of CSV records for importing. A package defines groups of records that are to be imported and the order they must be processed in. These groups are referred to as *segments*. A package can contain any number of segments. Each segment is considered to be an indivisible entity. The import system will import the whole of a segment or none of it. An error anywhere in the segment will cause the whole segment import to be discarded.

A package segment can either be self contained or it can refer to a separate CSV file. When self contained the actual CSV records are contained within the package file. Otherwise a reference is made to the CSV file (or part of it) that contains the records.

In summary: a package *controls* what records are to be imported and in what order.

1.4 What is a CSV file?

A *CSV File* is a text file that contains records, each of which consists of a set of fields in a known order with a known meaning. Each record is one line of text. Each field is separated from its neighbours by a designated character (the *CSV Separator*). The fields may also be quoted. A quoted field is one that has a leading and trailing quote character. These are usually discarded during the import process. The fields represent the information that is to be imported.

In summary: CSV records contain the *information* that is to be imported.

1.5 What is a Text file?

A *Text File* is similar to a CSV File except each line may contain a different set of fields and in different formats. In the context of the import system, a text file is usually a report produced by some other system and sent to a file instead of a printer. Typically, these reports contain useful information mixed in with formatting information, such as page headers and footers. Provided the useful information can be easily distinguished, the import system can extract the useful information and import it.

1.6 What is a group?

A *Text File* report is often structured into groups, with a header for each group, followed by a number of detail lines. For example, a bill of materials report may consist of a header to identify a part, followed by a line for each component required to make that part. The import system can be directed, via an *Import Filter*, to detect these groups and extract the detail information independently of the header information.

An example of this mechanism, based on the *Sage Line 50 Stock Explosion* report, is given later.

1.7 What should I be aware of?

In most cases, information to be imported will need processing by an *import filter* to make it acceptable. Typically, one CSV record will create several Match-ITrecords. The complete process of extracting information from one system, translating it, and importing it can be quite involved. There is plenty of scope to make a mistake!

If you discover this after importing thousands of records you might find it difficult to isolate and remove the erroneous records. To minimise this possibility we recommend that you try the process on just a few records first. Even safer, try it on a non live data set first, e.g. your training data.

Also, any omitted fields in the import records are automatically filled with whatever the current system default is set to. So, for example, if a material location is omitted then the default location will be assumed (normally

‘Stock Room’).

‘Stock Room’).

1.8 How do I create an import filter?

A 'filter' is a script file that contains a description of how to translate a CSV file, or any text file, into an equivalent import package.

Some example filter scripts are installed with Match-IT. They can be found in your **...\match_it\configs** folder.

One example has the name **sagestok.fil**. This example will translate the standard Sage Sterling stock file CSV report (**stock.csv**) into a form acceptable to Match-IT.

Another example is **sagebom.fil**. This will take the Sage Stock Explosion report and translate it into a form that adds the bill of materials information to the stock records created by the first example.

Note: The transformations possible using these import filters are very simple. When complex transformations are required, use a Lua script.

An *import filter* is a text file. You can create it using a text editor (such as Notepad or WordPad) or a word processor, providing you save the document in text form. When Match-IT is installed a suitable text editor, called *FoldIt*, is placed in your `...\match_it\sys` directory.

To be understood by the import filter the content of the filter file must conform to a *syntax*.

The required syntax is defined here in BNF (Backus-Naur Form):

```
CommentLine ::=       comment lines begin with '!' and can appear anywhere and are ignored
PackageGenerationScript ::=IssueLine [ NamesSpec ] { PacketSpec }
IssueLine ::=         'ISSUE:' SystemIssueId (just passed through to the package script)
NamesSpec ::=         'NAMES:'
                           { 'F#' Number '=' Name [ '!' description ]}
                      'ENDNAMES:'
                      Number is any positive integer. Name is an arbitrary string provided
                      it begins with a letter (a-z) and does not contain spaces or any of
                      the other special characters (?, !, =, [, ~, :). Description is
                      ignored and extends to the end of the line. The NAMES: section is
                      optional, if present it provides a means to give field numbers a
                      meaningful name. The use of F#Name or F#Number anywhere in the
                      script becomes synonymous.
PacketSpec ::=        'PACKET:' PacketName
                           [ StreamSpec ]
                           { LookupSpec }
                           { RecordSpec }
                      'ENDPACKET:'
PacketName ::=        an identifying name for the packet, eg. Sage Stock File
StreamSpec ::=        'STREAM:' SNumber
                           { IgnoreSpec }
                           { FieldSpec }
                           [ GroupSpec ]
                      'ENDSTREAM:'
IgnoreSpec ::=        'IGNORE:' Condition
GroupSpec ::=         'GROUP:' GNumber 'WHEN:' Condition {‘OR:’ Condition | ‘AND:’
                      Condition}
                           { CommentLine | IgnoreSpec}
                           { FieldSpec }
                      'ENDGROUP:' Condition {‘OR:’ Condition | ‘AND:’ Condition}
FieldSpec ::=         FixedField | CSVField
FixedField ::=        'F#' FNumber '[' FieldSize ']'
CSVField ::=          'F#' FNumber [',']
LookupSpec ::=        'LOOKUP:' LNumber 'IN:' (FileId|'SCRIPT') ['FIX:' FieldName]
                      ['WHEN:' Condition] {‘OR:’ Condition | ‘AND:’ Condition}
                           { FieldNameSpec }
                           { RecordImage }
                      'ENDLOOKUP:'
RecordSpec ::=        'RECORD:' (FileId|DropId) 'SEGMENT:' SegmentName [Options] ['WHEN:'
                      Condition] {‘OR:’ Condition | ‘AND:’ Condition}
                           { FieldNameSpec }
                           { RecordImage }
                      'ENDRECORD:'
FieldNameSpec ::=     'FIELDS:' FieldName { ',' FieldName }
FieldName ::=         the name of a field in the associated file. The order of the field
                      names given defines the order they are being given in the
                      RecordImage. This need not be the same as the physical file order.
                      See 'importing files'. In the context of a 'SCRIPT' LOOKUP: the
                      field name list is ignored.
FileId ::=            one of the system file TLAs
DropId ::=            one of the system DO operation names
SegmentName ::=       the name of the segment in the package script that the records are
                      to be generated into. If the segment does not exist, it is created.
                      The records generated are added to the end of the segment. A
                      'segment' refers to the records between a start and end pattern.
                      These are the patterns that will be passed to gxImportPackage. The
                      filter function creates start and end patterns and writes the
                      appropriate IMPORT script lines when it generates the translated
                      records. The start pattern is:
                      '{{{  (FileId|DropId):SegmentName:Part#'
                      and the end pattern is always '}}}'. I.e. plain-text FoldMarks.
                      (Part# is 1..N and is used to break up large segments)
Options ::=           options as required on the IMPORT line in an import package
Condition ::=         '[' FieldValue ']' WhenOpCode [ '[' MatchPattern ']' ]
FieldValue ::=        [FieldContext ':'] FieldRef
FieldRef ::=          'F#' FNumber [',' FieldSymbol] (the value of FieldSymbol in field
                      FNumber)
                      'R#' GNumber (the record number within the Group defined by GNumber)
                      (if no group is defined it means the overall record number)
                      'F#0' (the whole input record)
FieldContext ::=      'S#' SNumber  (field is in stream number SNumber)
                      'L#' LNumber  (field is in lookup number LNumber)
                      'L#0'         (field is a pre-defined value)
                      Omitted implies S#1.
FieldSymbol ::=       [ WordSeparator ] FieldSymbolNo
WordSeparator ::=     The character to consider as the element separator, omitted=comma
                      (,)
                      The ? ! ] - and space characters *cannot* be used. To use one of
                      these use Q for ?, E for !, C for ], M for - and S for space.
FieldSymbolNo ::=     MakePacked field element, first is 1. Omitted means all symbols.
                      A positive number means extract that symbol.
                      Eg. "Sterling,1.54" symbol 1 is "Sterling", symbol 2 is "1.54"
                      A negative number means strip up to and including that symbol and
                      return all remaining symbols.  Eg. For "X,Y,Z" symbol -1 is "Y,Z"
WhenOpCode ::=        '==' | '<>'  (equal, not equal)
                      '<<' | '>>'  (contains, does not contain)
                      '!<' | '>!'  (starts with, ends with)
                      '++'         (changed, the field value has changed from the previous
                      record, MatchPattern must be [])
MatchPattern ::=      Any string, case is not significant, Clarion '<##>' notation is
                      allowed.
                      Leading and trailing single quotes are ignored. This is useful for
                      patterns that      require leading or trailing spaces. E.g. ['  ']
                      is a string of 2 spaces, but [  ] is an empty string.
RecordImage ::=       { Text | FieldText }
FieldText ::=         '[' FieldValue FieldOpCode [ Text | '[' FieldValue ']' ] ']'
FieldOpCode ::=       '?' | '!' | '=[' text ']' | '~[' text ']'
                      (not blank and blank and equal a constant and not equal a constant)
                      (? = Text or FieldValue if FNumber has a non-blank and non-0 value,
                      else blank)
                      (! = Text or FieldValue if FNumber has a blank or 0 value, else
                      blank)
                      (=[text] = Text or FieldValue if FNumber has a value of 'text', else
                      blank)
                      (~[text] = Text or FieldValue if FNumber has a value that is NOT
                      'text', else blank)
                      (Note: [F#nnn?F#nnn] is equiv to [F#nnn] and F#nnn=[] is equiv to
                      F#nnn! And F#nnn~[] is equiv to F#nnn?)
```

```
(In a FieldText element, spaces are only allowed within the Text
portion)
```

The meaning of each of these syntax elements is:

PackageGeneration Script: A package generation script consists of an ISSUE line followed by any number of PacketSpecs.

CommentLine: Comments can be placed almost anywhere to aid the readability. They are ignored by the filter function.

IssueLine: The ISSUE line defines which issue of the Match-IT system the filter script is intended for. This is just passed through to the package script created by the filter.

PacketSpec: A packet spec consists of a PACKET line followed by a StreamSpec, followed by any number of record or lookup specs, followed by an ENDPACKET line. The entire packet spec is processed for each CSV record. Each record spec represents a potential Match-IT record that can be created from a CSV record.

PacketName: The packet name is used to identify which packet is to be processed when the filter script is run.

StreamSpec: A StreamSpec consists of a STREAM line followed by up to 16 IgnoreSpec lines, followed by a FieldSpec line, followed by an optional GroupSpec, followed by an ENDSTREAM line. A stream spec defines how to interpret the CSV Records and Text Records presented to the filter. Each IgnoreSpec identifies lines that should be ignored. The FieldSpec identifies the field boundaries within the record. The GroupSpec defines the Group structure (if present). The fields defined in the FieldSpec at this level define the group header fields. These are given field numbers from 1. These are prepended to each group line detected.

IgnoreSpec: An ignore spec defines a Condition for those records that should be ignored. If the condition is met the record is discarded and no information is extracted from it. There can be up to 16 ignore specs in total. Those defined before the GroupSpec apply only to the group header lines. Those defined within the group apply only to the group detail lines.

GroupSpec: A group spec consists of a GROUP line followed by up to 16 IgnoreSpec lines followed by a FieldSpec line, followed by an ENDGROUP line. A group spec defines how to interpret the lines within a Group. A Group is all the lines between, and including, the lines that meet the Condition on the GROUP line and the Condition on the ENDGROUP line. The fields defined in the FieldSpec at this level define the group detail fields. These are given field numbers that follow on from the group header fields defined in the StreamSpec. For each group line, the header fields are prepended. A typical use for this is to break up a file that contains sub-sections, e.g.

SubSection Header Line
    SubSection Detail Line 1
    SubSection Detail Line 2

    …
    SubSection Detail Line N

Can be broken up into:

SubSection Header Line SubSection Detail Line 1
SubSection Header Line SubSection Detail Line 2

…
SubSection Header Line SubSection Detail Line N

Note: The 'WHEN' condition of a group start is referencing stream fields, but the 'WHEN' condition of a group end is referencing group fields.

FieldSpec: A field spec defines the boundaries of a field in an input record. A field can be either a FixedField or a CSVField. The reference point for the start of each field is the end of the previous field. Thus: F#1,F#2[16] means field 1 extends to a separator, and field 2 is the next 16 characters. They must be defined in numerical order.

FixedField: A fixed field is a field that is defined by a fixed number of characters. The number between the square brackets [] defines the number of characters occupied by the field. The start of the field is defined as the end of the preceding field, irrespective of the type of the previous field. Thus: F#1,F#2[22] defines that field 2 is 22 characters following on from wherever field 1 stopped. Similarly, F#1[10]F#2[22] defines that field 2 starts at position 11 and occupies the next 22 positions.

CSVField: A CSV field is defined to extend from the end of the previous field to the next separator. Thus: F#1[10]F#2,F#3 defines that field 2 starts at position 11 and extends to the next separator, and field 3 starts from there, and so on.

LookupSpec: A LookupSpec consists of a LOOKUP line, followed by any number of FieldNameSpec lines, followed by any number of RecordImage lines, followed by an ENDLOOKUP line. A LookupSpec where a 'FileId' is given defines a record to be looked up in the Match-IT database. The RecordImage must load the appropriate key fields. There may be multiple lookups on the same file. Fields looked up may be referenced in the RECORD: clause using the 'L#' context notation. Fields are referenced in the order specified in the field name list, e.g. L#?:F#1 is the first L#?:F#2 is the second, etc. The first fields in the list must match some key in the file. If no FIX: clause is present, the LOOKUP is exact, i.e. an exact match must be found. If FIX: is present, the lookup will find an exact match for the fields up to and including the FIX: field, then the nearest match for the rest. (It uses a zfSet/zfNext pair.) Lookups are performed in the order they are defined. A failed lookup leaves all fields blank when referenced. All lookups are done in lookup number order and before any record image is created. The lookup is only performed if the WHEN: conditions are met. If the conditions are not met, all lookup fields are blank when referenced.

The lookup number must be 1 or greater. The lookup number 0 is a special case used to access pre-defined values. There is a fixed reportoire of pre-defined values. As of 30/01/03 this reportoire is:

F#1 = today's date
F#2 = the time now
F#3 = logged in user

A LookupSpec where the 'SCRIPT' option is given just evaluates fields for easier reference elsewhere. This is useful to minimise repetition and centralise 'knowledge'.

An example use might be to encapsulate things like: '[F#3][F#3![F#4]]'. These evaluations can also be cascaded, so they can be used to evaluate multiple choices.

Example:

```
LOOKUP:1 IN: SCRIPT
   [F#1=[X]Yes][F#1=[Y]No]
ENDLOOKUP:
```

!L#1:F#1 is now either 'Yes' or 'No' or blank.

```
LOOKUP:2 IN: SCRIPT
   [L#1:F#1][L#1:F#1!Illegal]
ENDLOOKUP:
```

!L#2:F#1 is now either 'Yes' or 'No' or 'Illegal'

RecordSpec:	A record spec consists of a RECORD line, followed by any number of FieldNameSpec lines, followed by any number of record images followed by an ENDRECORD line. Each record spec represents a record in one of the Match-IT files. A record spec is only considered if its 'WHEN:' conditions are TRUE. If no conditions are given then TRUE is assumed. The conditions are considered strictly in sequence. OR:'s are considered in turn, AND:s are considered until FALSE is detected. An FNumber is referring to the number of the field in the source record. The first is field 1. The whole input record is field 0.
FieldNameSpec:	A FieldNameSpec defines the logical order of the fields in a Match-IT database for the purposes of the import filter. It need not be the same as the physical order. Each name given must correspond to a name of a field in the associated file. The order the names are given defines the order they are being created in the RecordImage. See below for a full description of a field name spec. When the record is constructing a DO operation parameter list, the FieldNameSpec must still be present, with one name per parameter, but the actual names are arbitrary.
FileId:	A file ID is a short hand name for one of the Match-IT files. They consist of three letters. The list of valid file IDs is available within the on-line help system.
DropId	A drop Id is the name of a DO operation. The list of valid DO operations is available within the on-line help system.
SegmentName:	This is a name, for reference purposes within the script, of a group of related records. Every record created as a result of the associated record spec will be placed in the same segment. The import system will treat each segment as an indivisible unit for importing purposes. The whole segment or none of it will be imported.
Options:	These are just passed through to the package script. However, if the [DUPS] option is set, the filter will only add records to the associated segment if there isn't an identical record there already. The only significance of this is to save memory space while the filter script is being processed.
Condition:	A condition consists of a Field Value, an operation and a MatchPattern. If the match pattern matches the field value referenced according to the operation, the condition is TRUE. Otherwise it is FALSE. This is used to control records that are ignored, group boundaries and records created. The operation defines the test to perform.
FieldValue:	A field value is a reference to a field from the current input or a lookup, or a special value. The FieldContext defines where the field value is to come from. The FieldRef defines which field is being referenced. If the context is omitted, S#1 is assumed.
FieldRef:	A field ref. identifies which field in the context is being referenced. The first field in a record is field 1, the next field 2 and so on. The Fnumber indicates which field is being referenced. The [F#nnn] string is replaced by the current value of the field. If a FieldSymbol is present then only that element of the field is being referenced.
FieldContext:	This defines where the field is to come from. It can be either from the input stream, designated by 'S#', or a lookup, designated by 'L#'.

FieldSymbol: A field symbol identifies an element within the field. These elements correspond to the Match-IT *LType* elements. It is most often useful when taking looked-up fields apart, for example, extracting the units from a measure.

WhenOpCode: This defines the type of match to perform on the match pattern. The possibilities are:

'=='	the field value must be the same as the pattern
'<>'	the field value must be different to the pattern
'<<'	the field value must contain the pattern somewhere
'>>'	the field value must not contain the pattern anywhere
'!<'	the field value must start with the pattern
'>!'	the field value must end with the pattern.
'++'	the field value must have changed form the last record

MatchPattern: This is the pattern to match against the associated field value. The case of letters is ignored. Any leading and trailing single quote characters (') are removed. The MatchPattern for the '++' operation must be [].

RecordImage: The record image contains the fields of the associated record that are to be used to create, or lookup, a Match-IT file record. It consists of any number of Text strings or FieldText values. Each can either be placed in the image unconditionally or only when a specified field has a value or does not have a value. An implied package separator exists between each line of a record image. Thus:

```
[F#1]
[F#2]
```

is equivalent to:

```
[F#1] | [F#2]
```

The changes made when constructing the image are: to replace FieldNumber strings with the current value of the field; to remove leading spaces from the line; and to translate sub-strings of the form '<##>' into the ASCII char that has the code '##'.

FieldText: This represents a 'conditional' field. The first FieldValue defines the field whose value is to be checked. The FieldOpCode defines the type of check to apply. If the condition is 'TRUE' then the second FieldValue is placed in the record image, otherwise nothing is placed in the image.

FieldOpCode: This defines the type of check to perform for a conditional field. A '?' means the field condition is only 'TRUE' when the preceding FieldValue has a value. A '!' means the opposite. The condition is 'TRUE' only when the associated FieldValue does not have a value. An '=[text]' means the field condition is only TRUE when the preceding FieldValue has the value 'text', A '~[text]' means the opposite to '=', i.e. it's TRUE when the field is not equal to 'text'.

Refer to the examples, given later, to see how this syntax is used.

The overall processing loop of a filter is:

```
FOR each input record
   EXTRACT 'header' fields
   FOR each record that passes the stream ignore specs
     IF a group is defined
       SEARCH for group 'when' record
       EXTRACT 'detail' fields and append to the 'header' fields
       FOR each record that passes the group ignore specs and is not the endgroup 'when'
         FOR each lookup defined
           BUILD record image
           LOOKUP record
         ENDFOR
         FOR each record defined
           BUILD record image
           ADD record
         ENDFOR
       ENDFOR
     ELSE no group is defined
       FOR each lookup defined
         BUILD record image
         LOOKUP record
       ENDFOR
       FOR each record defined
         BUILD record image
         ADD record
       ENDFOR
     ENDIF
   ENDFOR
 ENDFOR
```

1.9 How do I create an import package?

A 'package' is a script file that contains a list of CSV file segments to be imported.

An *import package* is a text file. You can create it using a text editor (such as Notepad or WordPad) or a word processor, providing you save the document in text form. When Match-IT is installed a suitable text editor, called *FoldIt*, is placed in your `...\match_it\sys` directory.

To be understood by the import system the content of the package file must conform to a *syntax* . The required syntax is defined here in BNF (Backus-Naur Form):

```
PackageScript ::=        { CommentLine } IssueLine { CommentLine | ImportLine } EndLine

CommentLine ::=          any line that is not an issue or an import or an end
IssueLine ::=            'ISSUE:' SystemIssueId
ImportLine ::=           'IMPORT:' [ImportPath] ['FROM:' StartPattern] ['TO:' EndPattern]
                         'AS:' (FileId | DropId) Options
EndLine ::=              'END:'

SystemIssueId ::=        this is just information, it defines the system issue when the
                         package was built
ImportPath ::=           the path to the file containing the CSV records to be imported. If
                         omitted the records are assumed to be encapsulated in the script
                         file itself.
StartPattern ::=         a string used to mark the beginning of the section in the file to
                         be imported. If omitted the first line in the file is assumed. If
                         present the line immediately following the one containing this
                         pattern is used.
EndPattern ::=           a string used to mark the end of the section in the file to be
                         imported. If omitted the last line in the file is assumed. If
                         present the import stops at the line immediately preceeding the
                         one containing this.
FileId ::=               the TLA of a file being imported into
DropId ::=               the name of a DO operation to execute
Options ::=              [ IgnoreDuplicates ] [ UpdateDuplicates ] [ FieldListAtLine1 ]
IgnoreDuplicates ::=     '[DUPS]'
UpdateDuplicates ::=     '[OVER]'
FieldListAtLine1 ::=     '[USELINE1]'
```

The meaning of each of these syntax elements is:

PackageScript:	A package script consists of an ISSUE: line followed by any number of IMPORT: lines and an END: line. If the import CSV records are embedded in the package script, they follow the END: line. The script is interpreted in line by line order until the EndLine or the end of the file is reached.
Comment Line:	Comments can be placed almost anywhere to aid the readability. They are ignored by the import system.
IssueLine:	The issue line defines the issue of the Match-IT system when the import package was first created. The IssueLine must precede the first ImportLine.
ImportLine:	An import line defines a group of CSV records to import. The import path defines the file the records can be found in. If this is blank, the records are assumed to be in the script file itself (following the END line). The start and end pattern define which part of the file is to be imported. All records following the given start pattern and up to, but excluding, the end pattern are imported. The first line after the start pattern must be the field list unless the [USELINE1] option is given. If the AS: clause is a FileId, the records will be imported into that file. The field list must consist of field names in that file. If the AS: clause is a DropId each CSV record is passed to the named DO operation. In this case the CSV records must consist of the DO parameters in the appropriate order containing

fields of the appropriate type. The field list must still be present, with one name per parameter, but the actual names are arbitrary. Any output fields of the DO operation must also be present but the output is ignored.

The IMPORT lines are assumed to be presented in reverse dependence order. I.e. records with no dependants must be imported before the records that refer to them.

EndLine: This marks the end of the IMPORT lines in the package.

Options: If '[DUPS]' is present then the existence of a matching record is tolerated, no error is reported and the import proceeds as if it was added. A 'matching' record is one where <u>all</u> key fields are the same. NB: This does <u>not</u> necessarily mean the record is unique. This option is ignored when executing a DO operation.

If '[OVER]' is present then any existing matching record is overwritten, no error is reported and the import proceeds as if it was added. A 'matching' record is one where <u>all</u> key fields are the same. NB: This does <u>not</u> necessarily mean the record is unique. This option is ignored when executing a DO operation.

If '[USELINE1]' is present then the first line of the source is interpreted as containing the field name list. Otherwise, the line immediately following the start pattern is used.

The field name list defines the order of the fields to expect for the file. Each FieldName must match the name of a field in the file unless it is preceded by a '!'. If the field name is preceded by an '!' the corresponding column in the import records is ignored.

The FileId in the field name list must match the file being imported into.

Note: Many of the initial system options are imported using these packages. Look in your **...\match_it \configs** folder for files with a **.PAK** extension for examples.

1.10 How do I create a CSV file?

Records can be imported to any database from a CSV representation.

A *CSV file* is a text file. You can create it using a text editor (such as Notepad or WordPad) or a word processor, providing you save the document in text form. When Match-IT is installed a suitable text editor, called *FoldIt*, is placed in your `...\match_it\sys` directory.

In most cases the CSV files will be automatically created by the software system that is providing the information to be imported. Consult the documentation of that system to find out how to do this.

For the purposes of this discussion there are two types of CSV File. Those produced by your 'donor' system and those produced by the Match-IT import filter. The 'donor' CSV files are processed by the import filters to produce Match-IT CSV files. The syntax of these is slightly different. For completeness, the syntax of both is given here.

The 'donor' CSV file syntax is:

```
DonorCSVFile  ::= { DonorCSVRecord }
DonorCSVRecord      ::= DonorCSVField { CSVSeparator DonorCSVField }
DonorCSVField ::= QuotedField | NonQuotedField
QuotedField         ::= QuoteCharacter FieldText QuoteCharacter
NonQuotedField      ::= FieldText
QuoteCharacter      ::= any character defined by the Match-IT default
CSVSeparator  ::= any character defined by the Match-IT default
```

If a field is quoted, then quote or separator characters embedded in the field itself are ignored. Thus: "abc,def","ghi" is interpreted as two fields, one of 'abc,def' and another of 'ghi'. Similarly: "abc"def"ghi","jkl" is considered to be two fields, one of 'abc"def"ghi' and another of 'jkl'.

The Match-IT CSV file syntax is:

```
MatchITCSVFile      ::= MatchITFieldList { MatchITCSVRecord }
MatchITFieldList    ::= MatchITFileId ':' MatchITFieldSpec { ',' MatchITFieldSpec }
MatchITFIleId ::= the three character mnemonic of the file the records are destined for
MatchITFieldSpec    ::= [ IgnoreField ] MatchITFieldName [ '[AS:' TypeName ']' ]
IgnoreField         ::= '!'
MatchITFieldName    ::= the name of a field in the file
TypeName            ::= this must match a logical type name, e.g. 'Material', 'Measure',
etc.
MatchITCSVRecord    ::= MatchITCSVField { FieldSeparator MatchITCSVField }
MatchITCSVField     ::= MatchITCSVElement { ElementSeparator MatchITCSVElement }
MatchITCSVElement   ::= a string that represents a value in the Match-IT file system
ElementSeparator    ::= this is always a comma (,)
FieldSeparator      ::= any character defined by the Match-IT package separator default
```

If `IgnoreField` is present, the field spec is ignored and the corresponding column in the CSV records is also ignored, this is intended to quickly turn a field 'off'.

In a Match-IT CSV file, fields may be composed of several elements. For example, a customer contact is defined by the customer name followed by the contact name. The elements required for each Match-IT field are defined elsewhere.

The number of elements in the MatchITFieldSpec must match that expected for the LType involved. E.g. "Each,1" for a measure.

If the [AS:...] clause is present it overrides the natural type designation for the field and interprets the field elements as the type defined by TypeName. This is intended to be used where the natural type is ambiguous. E.g. a 'MatStep' and a 'PrcStep' are both MCB references but one is defined via an MCH and the other via an MCD. So for the mch:Step field (a [KEYOF:mcb]), a field spec of Step[AS:MatStep] would expect a string of the form 'MaterialName,StepName' to yield an MCB ref for a material method, and a field spec of Step [AS:PrcStep] would expect a string of the form 'ProcessName,StepName' to yield an MCB ref for a process method.

Note: Importing like this is a crude field-to-field copy. If the data needs processing in any way, it's usually better to write a Lua script.

1.11 How do I create a report text file?

You can create it using a text editor (such as Notepad or WordPad) or a word processor, providing you save the document in text form. When Match-IT is installed a suitable text editor, called *FoldIt*, is placed in your `...\match_it\sys` directory.

In most cases the report text files will be automatically created by the software system that is providing the information to be imported. Consult the documentation of that system to find out how to do this.

For completeness the syntax of a typical report text file is given here:

```
ReportTextFile      ::= { Page }
Page                ::= PageHeader { Group } PageFooter
Group               ::= GroupHeader { GroupDetail } GroupFooter
GroupDetail         ::= { Field }
```

1.12 How do I run an import filter?

This is done from the *Import File(s)* form. A way to reach this form using the standard menus is: `Favourites | Functions | Maintenance | Import File(s)`. Just select the required import method, fill in the appropriate fields and press the **`Import`** button.

1.13 What can be imported?

For all practical purposes, records for any Match-IT file can be imported. The list of files and their fields is available from the on-line help system.

The help topics for a file definition show the field name as tla:Name, followed by a brief description. The tla is the 3 character ID for the file, and the Name is the field name as it should be used in the import system.

LType Elements

LType is the term used to refer to the Match-IT data types that are expected in the import fields. These may be single or multi element text strings. The list of these types and the text elements required is available in the on-line help system.

1.14 A CSV example: Sage Line 50 Stock File

The system installs with an example import filter for the Sage Line 50 stock file. The example is given here along with the package script it generates. Also shown are a few sample 'donor' CSV records that were used to create the sample package script.

Filter Script

```
ISSUE: 1.18

PACKET: Sage Stock File
  !Sage stock format (standard stock.csv)
  ! F#1 Stock code         -->mch:Material
  ! F#2 Stock description  -->mch:Name
  ! F#3 Category no.       -->(create a Class kode) mch:Class
  ! F#4 Department         -->NA
  ! F#5 Nominal code       -->mch:SellingNomCode and mch:BuyingNomCode
  ! F#6 Supplier ref.      -->(create a CSH record)(create a MCS record)mcs:Supplier
  ! F#7 Unit of sales      -->(create a UMT)mch:PackQty,mch:AllocationUnit,mch:ManQty
  ! F#8 Sale price         -->mch:PackPrice
  ! F#9 Tax code           -->NA
  ! F#10 Discount A        -->(create a MCQ)mcq:Discount
  ! F#11 Discount B        -->NA
  ! F#12 Discount C        -->NA
  ! F#13 Re-order level    -->NA
  ! F#14 Re-order qty      -->mch:ReOrderQty
  ! F#15 Part reference    -->mcs:TheirPartNum
  ! F#16 Location          -->(create Location kode)mch:HomeLocation
  ! F#17 Comodity code     -->NA
  STREAM:
    IGNORE: [F#0] == []
    F#1,F#2,F#3,F#4,F#5,F#6,F#7,F#8,F#9,F#10,F#11,F#12,F#13,F#14,F#15,F#16,F#17,
  ENDSTREAM:

  RECORD:umt SEGMENT:StockUnits [DUPS] WHEN: [F#7] <> []
  FIELDS:Unit,Class,Name,Picture,Description
    [F#7]|General|[F#7]|@n-24.2~ [F#7]s~|Imported from Sage Stock File
  ENDRECORD:

  RECORD:kdt SEGMENT:StockClasses [DUPS] WHEN: [F#3] <> []
  FIELDS:Class,Code,Name,Protected,Description
    StockClass|[F#3]|Category [F#3]||Imported from Sage Stock File
  ENDRECORD:

  RECORD:kdt SEGMENT:Locations [DUPS] WHEN: [F#16] <> []
  FIELDS:Class,Code,Name,Protected,Description
    Location|[F#16]|[F#16]||Imported from Sage Stock File
  ENDRECORD:

  RECORD:mch SEGMENT:StockRecords
  FIELDS:Material,Name,Class,Group,Family,Standard,Sellable,Buyable,PackQty,PackPrice
  FIELDS:ExtraCost,ExtraSetupCost,ManQty,IsModule,ReviewOn,RejectRate,ReOrderQty,HomeLocat
ion
  FIELDS:JobCardLayout,SpecSheetLayout,DeviceLabelLayout,CofCLayout,SubContractLayout
  FIELDS:BuyingNomCode,SellingNomCode,Length,Width,MinUseQty,MinUseLength,MinUseWidth
  FIELDS:ExtraSetupTime,ExtraCycleTime,CutWidth,Squaring,MinBatchSize,MaxBatchSize,Descrip
tion
    [F#1]|[F#2]|[F#3?StockClass,][F#3]||||Yes|Yes|[F#7][F#7?,1]|[F#8?Sterling,][F#8]
    ||[F#7][F#7?,1]|Yes|||[F#7!Each][F#7],[F#14!1][F#14]|[F#16?Location,][F#16]
    ||||
    [F#5]|[F#5]|||[F#7!Each][F#7],0||||||1|1000000|Imported from Sage Stock File
  ENDRECORD:
ENDPACKET:
```

Package Script

```
!  Package script created by gxImportFilter
!
!          Created:  9/12/96 at 22:10:56
!    Filter Script: F:\MATCH_IT\CODE_DEV\GX\GX_SSTOK.FIL
!           Packet:
!       Source CSV: E:\CLIENTS\K2\SAGE\STOCK.CSV
!         Begin At:
!           End At: ACETAL.8MM
!     Package file: g:\temp\test.pak
!    CSV Separator: ,
!     Strip quotes: 1
!Package separator: |

ISSUE: 1.17

IMPORT: FROM:{{{  umt:StockUnits:1                    TO:}}} AS:umt [DUPS]
IMPORT: FROM:{{{  kdt:StockClasses:1                  TO:}}} AS:kdt [DUPS]
IMPORT: FROM:{{{  mch:StockRecords:1                  TO:}}} AS:mch

END:

{{{  kdt:StockClasses:1
kdt:Class,Code,Name,Protected,Description
StockClass|16|Category 16||Imported from Sage Stock File
}}}
{{{  umt:StockUnits:1
umt:Unit,Class,Name,Picture,Description
LOT|General|LOT|@n-24.2~ LOTs~|Imported from Sage Stock File
OFF|General|OFF|@n-24.2~ OFFs~|Imported from Sage Stock File
}}}

{{{  mch:StockRecords:1
mch:Material,Name,Class,Group,Family,Standard,Sellable,Buyable,PackQty,PackPrice,ExtraCost
,ExtraSetupCost,ManQty,IsModule,ReviewOn,RejectRate,ReOrderQty,HomeLocation,JobCardLayout,
SpecSheetLayout,DeviceLabelLayout,CofCLayout,SubContractLayout,BuyingNomCode,SellingNomCod
e,Length,Width,MinUseQty,MinUseLength,MinUseWidth,ExtraSetupTime,ExtraCycleTime,CutWidth,S
quaring,MinBatchSize,MaxBatchSize,Description

1SIBIT.00001|O RING 25MM ID 5MM CROSS SECT.||StockClass,16|||LOT,1|Yes|Yes|LOT,1|
Sterling,20.00|||||LOT,1|Yes|||||LOT,1.00|||||||4000|4000|Imported from Sage Stock File

1SIBIT.00002|19MM DIA 2MM S/A EPDM WASHER||StockClass,16|||OFF,1|Yes|Yes|OFF,1||||||OFF,1|
Yes|||||OFF,360.00|||||||4000|4000|Imported from Sage Stock File
}}}
```

Donor CSV Samples

```
"1SIBIT.00001    ","O RING 25MM ID 5MM CROSS SECT.", 16,  3,"4000  ","      ","LOT     ",
  20.00,"T1",  0.00,  0.00,  0.00,    0.00,    1.00,"                ","                "

"1SIBIT.00002    ","19MM DIA 2MM S/A EPDM WASHER  ", 16,  2,"4000  ","      ","OFF     ",
   0.00,"T1",  0.00,  0.00,  0.00,    0.00,  360.00,"                ","                "
```

1.15 A Text File example: Sage Sterling Stock Explosion

The Match-IT system installs with an example import filter for the Sage Line 50 Stock Explosion report. This filter extracts the bill of materials information. The example is given here along with the package script it generates. Also shown are a few sample 'donor' records that were used to create the sample package script.

Filter Script

```
ISSUE: 1.18

PACKET: Sage Stock Explosion Report

  !Input to STREAM: is Sage Stock Explosion report sent to a file
  !Output from STREAM: is
  !  F#1=Assembly stock code
  !  F#2=Part stock code
  !  F#3=junk
  !  F#4=Part quantity
  STREAM:1
    IGNORE: [F#0] == []
    IGNORE: [F#0] << [<12>]
    IGNORE: [F#0] << [<27>]
    IGNORE: [F#0] << [Date :]
    IGNORE: [F#0] << [Page :]
    IGNORE: [F#0] << [Printed:]
    F#1[22]
    GROUP:1 WHEN: [F#0] !< [----]
      IGNORE: [F#0] !< [----]
      IGNORE: [F#0] !< ['  ']
      IGNORE: [F#0] << [<12>]
      IGNORE: [F#0] << [<27>]
      IGNORE: [F#0] << [Date :]
      IGNORE: [F#0] << [Page :]
      IGNORE: [F#0] << [Printed:]
      F#2[22]F#3[77]F#4[8]
    ENDGROUP: [F#0] == []
  ENDSTREAM:

!Load mch:Material field with Part stock code and get record
  LOOKUP:1 IN:mch
  FIELDS:Material,ManQty
    [S#1:F#2]
    !Part stock code read from the input stream
  ENDLOOKUP:

  RECORD:mcg SEGMENT:Parts
  FIELDS:Assembly,Rank,PartName
    [F#1]|[R#1]|[F#1]:Part#[R#1]
  ENDRECORD:

  RECORD:mca SEGMENT:Materials
  FIELDS:Component,Rank,Material,Quantity,Length,Width
    [F#1],[R#1]|[R#1]|[F#2]|[L#1:F#2,1],[F#4]
    ![L#1:F#2,1] is the units name from mch:ManQty looked up
  ENDRECORD:

ENDPACKET:
```

Package Script

```
!  Package script created by gxImportFilter
!
!          Created: 23/03/97 at 18:51:10
!    Filter Script: E:\CLIENTS\BBRIDGE\MATCH_IT\DATA\LEGACY\5_BOM.FIL
!           Packet:
!       Source CSV: G:\TEMP\TEST.BOM
!         Begin At:
!           End At:
!     Package file: G:\TEMP\TEST.PAK
!    CSV Separator: ,
!           Quotes: '"`
!Package separator: |

ISSUE: 1.18

IMPORT: FROM:{{{  mcg:Parts:1                        TO:}}} AS:mcg
IMPORT: FROM:{{{  mca:Materials:1                    TO:}}} AS:mca

END:


{{{  mca:Materials:1
mca:Component,Rank,Material,Quantity,Length,Width
|00244/1,1|1|11-700P-W|Each,    1.00
|00244/1,2|2|SC-UCC11-226-AAR|Each,    1.00
|00696/1,1|1|543-282|Each,    1.00
|00696/1,2|2|554815-1|Each,    1.00
|00696/1,3|3|554831-1|Each,    1.00
|00696/1,4|4|6-700P-W|Each,    1.00
|00696/1,5|5|CW13083PAIR|Metre,   30.00
|00696/1,6|6|H30X30MMBLACK|Each,    1.00
|00714/1,1|1|11-700P-W|Each,    1.00
|00714/1,2|2|SC-UCC11-226-CKO|Each,    1.00
|01205/1,1|1|001677261001|Metre,   15.00
|01205/1,2|2|01205SUBASSY|Each,    1.00
|01205/1,3|3|6-700P-W|Each,    3.00
|01205/1,4|4|65153-001|Each,    2.00
|01205/1,5|5|65155-001|Each,    3.00
|01205/1,6|6|75885-001|Each,    2.00
|01205/1,7|7|ATUM19/6BLACK-L|Metre,    0.10
|01205/1,8|8|BI1006|Each,    6.00
|01205/1,9|9|BI1007|Each,    3.00
|01205/1,10|10|XY80|Metre,    1.57
}}}
{{{  mcg:Parts:1
mcg:Assembly,Rank,PartName
00244/1|1|00244/1:Part#1
00244/1|2|00244/1:Part#2
00696/1|1|00696/1:Part#1
00696/1|2|00696/1:Part#2
00696/1|3|00696/1:Part#3
00696/1|4|00696/1:Part#4
00696/1|5|00696/1:Part#5
00696/1|6|00696/1:Part#6
00714/1|1|00714/1:Part#1
00714/1|2|00714/1:Part#2
01205/1|1|01205/1:Part#1
01205/1|2|01205/1:Part#2
01205/1|3|01205/1:Part#3
01205/1|4|01205/1:Part#4
01205/1|5|01205/1:Part#5
01205/1|6|01205/1:Part#6
01205/1|7|01205/1:Part#7
01205/1|8|01205/1:Part#8
01205/1|9|01205/1:Part#9
01205/1|10|01205/1:Part#10
}}}
```

Donor Text File

```
XYZ LIMITED                    Stock Reports - Stock Explosion                    Date :
270297
                                                                                  Page :
1
                                  Printed: 19:39 27-02-97


00244/1             50WAY BT226 CABLE ASSY.(Iss.1)        Last Cost Price  Location
Quantity
------------------------------------------------------------------------------------------
---
11-700P-W           *WHITE PAPER ADHESIVE LABEL              0.01  OFFICE CUPBOARD
1.00
SC-UCC11-226-AAR    226 CAB.ASSY ISSUE 1. (00244)           13.86
1.00

00696/1             CABLE ASSY P/No: 00696 ISSUE 1        Last Cost Price  Location
Quantity
------------------------------------------------------------------------------------------
---
543-282             SLEEVED CABLE GROMMET                    0.02  AZ04
1.00
554815-1            CONNECTOR 24WAY BACK/BACK AMP            4.43  AD02
1.00
554831-1            COVER KIT 24WAY BACK/BACK AMP            1.22  AD03
1.00
6-700P-W            *WHITE PAPER ADHESIVE LABEL              0.01  OFFICE CUPBOARD
1.00
CW13083PAIR         *CW1308 3 PAIR CABLE                     0.07  BN01
30.00
H30X30MMBLACK       BLACK NEOPRENE SLEEVE 30mmLONG           0.03  AZ03
1.00

00714/1             CABLE ASSY P/No: 00714 ISSUE 1        Last Cost Price  Location
Quantity
------------------------------------------------------------------------------------------
---
11-700P-W           *WHITE PAPER ADHESIVE LABEL              0.01  OFFICE CUPBOARD
1.00
SC-UCC11-226-CKO    50WAY CHAMP ASSY.(150Ft) Iss.1           35.57  AA01
1.00

01205/1             CABLE ASSY P/No 01205 ISSUE 1.        Last Cost Price  Location
Quantity
------------------------------------------------------------------------------------------
---
001677261001        B/PAN 50x2xAWG26(1)UNS CODE7             3.60  BB03
15.00
01205SUBASSY        SUB KIT FOR MERCURY 01205                0.00
1.00
  67023-001           3x32 26AWG(1) PIERCE BLOCK             0.38  AG04
1.00
  77345-411           3x32 26AWG (1) CONTACT BLOCK           5.50  AG05
1.00
6-700P-W            *WHITE PAPER ADHESIVE LABEL              0.01  OFFICE CUPBOARD
3.00
65153-001           32x2 PIERCE BLOCK                        0.36  AH04
2.00
65155-001           DIN41612 BLACK UNIVERSAL COVER           0.40  AH02
3.00
75885-001           2x32 (26AWG(1) GOLD CONT BLOCK           8.10  AI04
2.00
```

```
ATUM19/6BLACK-L     ATUM 19/6mm BLACK H/SHRINK(24M          4.85  BK02
0.10
BI1006              *No.2x3/8"PAN/SLOT 76412-002            0.01  AE01
6.00
BI1007              CABLE TIE PANDUIT SST 1M-M              0.02  AE01
3.00
XY80                8mm PVC BLACK HELLERMAN SLEEVE          0.09  BM04
1.57
```

2 Import File Include Sections

The source file scanning system implements a mechanism to allow one file to reference another. All lines in the referenced (section of) the file appear as if they were defined within the enclosing file. Two directives are recognised by zbScanNext to implement this. They are:

1) !#INCLUDE('FileName'[,'SectionName'])

and

2) !#SECTION('SectionName')

These directives must be at the start of the line. The semantics mimic the Clarion compiler INCLUDE and SECTION directives. In brief: in an !#INCLUDE directive the contents of the file named by 'FileName' is inserted at the point of the !#INCLUDE directive. If a 'SectionName' is given, only lines following the line starting with !#SECTION('SectionName'), and up to the next !#SECTION, or the end of file, are inserted. If the 'FileName' is a relative file name, it is relative to the enclosing file. The 'FileName' and 'SectionName' **must** be enclosed in single quote marks (') with no leading spaces.

3 Security Scheme Import Syntax

A security scheme import file is a text file that can contain leading comment lines and a series of [GROUP] 'folds'. A 'fold' is a set of lines preceded by a line of "{{{ 'fold title'" and terminated by a line of "}}}". By convention, these files have a `.SEC` file extension. The default security scheme can be seen in the `configs` folder and provide comprehensive examples.

... [GROUP]

The syntax of the fold title is one of:

- [GROUP] GroupNumber [IS] GroupName
- [GROUP] GroupNumber [DO]
- [GROUP] ALL

When GroupNumber is in the range 2..30 it identifies the group being defined. When GroupNumber is ALL, the settings in the fold are applied to all groups (2..30). Group 1 (the quickSOFT group) cannot be changed. When the [IS] option is present, the GroupName is the name to give the group and its description is reset (so the group fold contents define it completely). When the [DO] option is present the existing group name and description are preserved. The contents of the fold define the access privileges available to the group. Only groups whose access is beyond the 'raw' defaults need be defined. Only groups the logged in user is a member of can be changed, and only things the user has access to can be changed.

Attempts to change things the user has no access to are (silently) ignored.

The 'raw' defaults are:

- (1) quickSOFT: everything
- (2) Agent: everything
- (3) System Admin: everything
- (4..30) Others: nothing

The fold can contain group description lines followed by any number of 'include' and 'exclude' lines. Facilities can be included/excluded at various levels:

- SYSTEM - refers to everything in the system
- MODULE - refers to everything in a module
- PROCESS - refers to a menu runnable process (ASP)
- FILE - refers to a file (ASD)
- FIELD - refers to a field (ASF)
- DEFAULT - refers to a default (VDS)
- WIZARD - refers to a wizard (DRH)
- MENU - refers to a menu (BMN)

The include/exclude lines are processed in the order they are encountered. The possible line types and their syntax/semantics are:

EXCLUDE_SYSTEM:flags	all group bits in all ASD, ASF, ASP, BMN, VDS and DRH records are turned off as directed by the flags
INCLUDE_MODULE:xx{,xx}	xx{,xx} is a comma separated list of module IDs
	all ASD, ASF, ASP, BMN, VDS, DRH records for the modules identified are turned on
INCLUDE_MODULE:xx{,xx}	xx{,xx} is a comma separated list of module IDs
	all ASD, ASF, ASP, BMN, VDS, DRH records for the modules identified are turned off
INCLUDE_PROCESS:p{,p}	p{,p} is a comma separated list of menu runnable processes
	all identified ASP records are turned on
EXCLUDE_PROCESS:p{,p}	p{,p} is a comma separated list of menu runnable processes
	all identified ASP records are turned off

INCLUDE_DEFAULT:d{,d}	d{,d} is a comma separated list of system default names
	all identified defaults are turned on
EXCLUDE_DEFAULT:d{,d}	d{,d} is a comma separated list of system default names
	all identified defaults are turned off
INCLUDE_WIZARD:w{,w}	w{,w} is a comma separated list of wizard names
	all identified wizards are turned on
EXCLUDE_WIZARD:w{,w}	w{,w} is a comma separated list of wizard names
	all identified wizards are turned off
INCLUDE_FILES:tla{,tla}	tla{,tla} is a comma separated list of file privileges
	all identified ASD records for the identified access are turned on
EXCLUDE_FILES:tla{,tla}	tla{,tla} is a comma separated list of file privileges
	all identified ASD records for the identified access are turned off
INCLUDE_FIELD:name{,name}	name{,name} is a comma separated list of field privileges
	all identified ASF records for the identified access are turned on
EXCLUDE_FIELD:name{,name}	name{,name} is a comma separated list of field privileges
	all identified ASF records for the identified access are turned off
INCLUDE_MENU:m{,m}	m{,m} is a comma separated list of menu names
	all identified BMN records are turned on
EXCLUDE_MENU:m{,m}	m{,m} is a comma separated list of menu names
	all identified BMN records are turned off
the flags syntax is: [P][F][S][V][D][W][M]	P=processes,F=files(full access),S=Files(select only),V=fields,D=defaults,W=wizards,M=menus
	if flags is not present PFVDWM is assumed (i.e. all present), if present they control what is to be included/excluded
the module ID syntax is: xx([P][F][S][V][D][W][M])	The xx is the two letter module mnemonic.
	The letters in []'s have the same meaning as the flags (above).
	xx==xx()==xx(PFVDWM)
the tla syntax is: tla([S][C][I][D])	The tla is the three letter file mnemonic.
	S=select,C=change,I=insert,D=delete
	tla==tla()==tla(SCID)
the field name syntax is: name([V][E][#])	V=view, E=edit, #=level (0..9)
	name==name()==name(VE0)

Note: The import file is read via the ZB source file scanning system. This provides an 'include' capability. Lines beginning !#INCLUDE and !#SECTION implement this and are recognised by ZB. Thus these lines can appear anywhere in the script. See the Import File Include Sections for details.

4 Menu Import Syntax

A menu import file is a text file that describes the required menu structure as a series of **FoldIt** (a simple text editor installed with Match-IT) folds. A 'fold' is a set of lines preceded by a line of "{{{ 'fold title'" and terminated by a line of "}}}". By convention, these files have a `.MNU` file extension. The default system menus can be seen in the `configs` folder and provide comprehensive examples.

Lines beginning with an '!' are ignored as are all lines up to the first fold line containing '[RIBBON]'.

There are five types of fold used:

- ... [RIBBON]
- ... [BUTTON]
- ... [GAP]
- ... [MENU]
- ... [MESSAGE]

... [RIBBON]

The syntax of the fold title is:

```
    [RIBBON] ButtonText [IS] ButtonHint [AS] RibbonName [COLOUR] ColourNumber
    [,ColourNumber] [FOR] UserGroups
```

A [RIBBON] fold will create a new ribbon menu according to its contents. The [AS] component defines the name of the ribbon in the BMN file. If RibbonName is omitted then ButtonText is used instead. If a BMN with the name RibbonName, or ButtonText, does not already exist then a new BMN record is created. The [COLOUR] component defines the background colour to use for the ribbon and its processes. They're represented as hexadecimal numbers (e.g. 00FFh). If omitted or 'NONE' the default colours are used. The [FOR] component defines the user groups allowed to run the ribbon as a decimal number representing the 'or' of all the mask bits involved. E.g. 7 == group 1, 2 and 3. If omitted the current user group membership is used. Groups 1,2,3 are always included. Except for the outermost ribbon, a BMI record is created that is an item in the immediately enclosing [RIBBON] fold's BMN. The ButtonHint will be put in that BMI record. The lines within the fold, up to the first contained fold, are the ribbon description. Lines are appended unless it's just a '.', in which case a line break is inserted. This is put in the BMN record and a quick help page with the same name as the ribbon. Folds inside the [RIBBON] fold become buttons on the ribbon.

... [BUTTON]

The syntax of the fold title is:

```
    [BUTTON] ButtonText [IS] ProcessName [WITH] P1 [AND] P2
```

This is defining a Match-IT process to run. The ButtonText appears on the button face. The ProcessName must be the name of some ASP record. A BMI record is created that references the Process. The BMI will be an item in the immediately enclosing [RIBBON] fold's BMN. The first line of text inside a [BUTTON] fold is the button hint. The rest is ignored. If there is no text then the button hint will be that of the process being referenced. The [WITH] P1, if present, defines the LType and value for the first BMI parameter. The [AND] P2, if present, defines the LType and value for the second BMI parameter. In both cases, the LType name precedes the value and is separated by a comma, e.g. `Template,~BookCase`

... [GAP]

The syntax of the fold title is:

```
    [GAP]
```

The fold contents are ignored. This creates a BMI in the immediately enclosing [RIBBON] fold's BMN that is not a process. It just creates a visual gap in the menu.

... [MENU]

The syntax of the fold title is:

```
    [MENU] ButtonText [IS] MenuName
```

This creates a button that calls up an existing menu. The ButtonText appears on the button face. The MenuName must be the name of some BMN record. A BMI record is created that references the Menu. The

BMI will be an item in the immediately enclosing [RIBBON] fold's BMN. The first line of text inside a [MENU] fold is the button hint. The rest is ignored. If there is no text then the button hint will be that of the menu being referenced.

... [MESSAGE]

The syntax of the fold title is:

```
[MESSAGE] ButtonText
```

The ButtonText appears on the button face. A BMI record is created. The BMI will be an item in the immediately enclosing [RIBBON] fold's BMN. The first line of text inside a [MESSAGE] fold is the button hint. The rest is shown as a message when the button is pressed. Lines in the fold are appended unless it's just a '.', in which case a line break is inserted.

Note:

The import file is read via the ZB source file scanning system. This provides an 'include' capability. Lines beginning `!#INCLUDE` and `!#SECTION` implement this and are recognised by ZB. Thus these lines can appear anywhere in the script. See the Import File Include Sections for details.

5 INI Files

Match-IT uses three types of 'INI' file to hold critical initialisation information. The *Local INI File* holds configuration information specific to a user. The *Global INI File* holds information relevant a specific dataset. The *System INI File* holds information relevant to a specific installation.

The contents of these INI files can be viewed by clicking on the splash screen that appears when first starting the system, then selecting the `Advanced Switches` tab on the `Set Command Line Options` dialog that appears. In the `Advanced Switches` tab select the `ini=` sub-tab. Four INI files can be accessed via the **`Edit ... file`** buttons on the bottom of this sub-tab.

INI files are simple text files with section names enclosed in square brackets [] followed by any number of items in the section consisting of lines of the form `name=value`.

Local INI File

The settings in this file consist of only those things necessary to allow Match-IT to start. For example, the location of the file registry, the dataset you are using, your login ID, etc. You would not normally modify any of the settings in here. If you delete the file completely Match-IT will re-create it the next time it starts. In so doing, it will ask you again for the necessary information.

The 'local' INI file is created on the station you are working at in your documents and settings folder under application data. If you are using terminal services to run the system remotely the station will be your local access station and not the remote terminal services server. The location of the application data folder is Windows version specific. The file name in that folder is typically `match_it.set` but this can be overridden by switches on the start-up icon.

Global INI File

The main settings in this file specify the name of the dataset and its version. It also specifies the parameters for the service that controls the 'agent'. All these settings are maintained automatically. If you delete this file it will be re-created. The global INI file resides in the same folder as the dataset it controls and is named `match_it.set`, typically `c:\match_it\data\match_it.set`

System INI File

The main purpose of the system INI file is to control access to the system during (re-)installation and upgrading. It typically only contains the [Exclusive] section which contains the name of the station the system is locked to for maintenance purposes. Blank means it's not locked. If you delete this file it will be re-created. The system INI file resides in the same folder as the system programs it controls and is named system`.set`, typically `c:\match_it\sys\system.set`

Agent INI File

The special 'agent' session also has it's own 'local' INI file, but it's called `agent.set` and resides in the dataset folder of the dataset the agent is controlling, typically `c:\match_it\data\agent.set`