NBN Record Cleaner Verification Rules

V1.0.8.3

August 2012

Stuart Ball, JNCC

Graham French, NBN Trust

Contents

1. Rule files

Rules are simple ASCII text files using the INI format. They can be edited with a text editor.

Verification rules are specified in simple text files which are located in subdirectories of the VerificationData directory. Since a separate file is needed for each rule to apply to each species, there will potentially be large numbers of rule files. It therefore makes sense to organise them in **rule-sets**. A group of rule files (usually with a common theme – e.g. the ranges of dragonflies) are zipped into a single file for delivery and are normally installed in their own sub-directory.

1.1. Creating rule files

Verification rules are intended to be created and maintained by expert groups, such as Recording Schemes and Societies, and made available via the internet. The rule-sets that a user chooses to use are downloaded to their computer for performance reasons. The application (providing it can connect to the internet!) performs a check at start-up for updates and/or new rule-sets and notifies the user if any such changes are available.

You can also design you own verification rules by writing suitable text files and placing them in the \VerificationData\Personal folder. This is especially useful for developing and testing rule files which will later be supplied via the internet mechanism, but can also allow custom rules to be implemented by an individual.

1.2. Scope

A rule can apply tests to data within one or more fields of a single record.

The rule may require the data to pass or fail the test it specifies (e.g. check whether the spatial reference is within the coastal boundary for a terrestrial species or is not within it for a marine species). The following types of rules are supported:

- **List membership**. Check that a given data item is included (or not included) in a given list. E.g. check the species is a dragonfly.
- **Period**. Check that the date given in a record is before and/or after a given date. It is possible to give dates with no year in order to test whether the date is within a given season in any year.

E.g. check that records of a migratory bird are between the expected arrival and departure dates for that species.

- **Geographical**. Check that the spatial reference given in the record is within (or outside) a given polygon or list of grid squares and/or vice-counties. E.g. check whether the record is within the known range of the species.

- **Regular expression**. Apply a given regular expression to a data item. E.g. check whether the entry in a "PostCode" field follows the rules for a valid post code.

In addition, **script files** allow custom external modules to be automatically imported into the system and offered as tests. These are written in un-compiled VB.NET code and allow for complex, custom tests to be implemented. For the vast majority of uses, it is not anticipated that external script modules will be needed. Scripts are supported for the rare occasions when an additional test is required that is not achievable in any other way. This is most likely to be the case if a test requires data from a combination of two or more columns (the standard test types mostly apply tests to data in a single column).

This should allow any conceivable test to be written - with one important limitation: **tests can only be applied to the data fields within a single record**. It is not possible to write rules that require data from two or more different rows of data. (An example that would require access to multiple rows would be a test for duplicate records.)

The final type of "rule file" is a little different in purpose, but makes uses of the same text file format and delivery mechanism. These are "**identification difficulty files**" which allow species to be tagged with a code to indicate difficulties with names or identification. These files do not specify pass/fail type tests, but allow rows in the verification result grid to be colour coded and provide a tool-tip message to the user when they mouse-over the coloured area. The intention is to bring records of "difficult species" to the attention of the user. These might include species that are difficult to identify correctly or where there is a problem with the name, such as a recent split or where confusion is likely due to synonymy.

1.3. Examples

An example format for each rule type is provided in the grey text boxes. Additionally rules used by the Record Cleaner are stored in the VerificationData folder on your computer. Examination of the rule text files for a particular test will give you a guide as to the required format for that particular rule type.

2. Supplying rules

Once you have created the rule files as outlined in sections 3 and 4 the next step is to supply them to NBN so that they can be available to use by any user of the NBN Record Cleaner Tool. Prior to submitting your rules you can test them in your local copy of NBN Record Cleaner by placing the rule files in VerificationData\Personal folder. Remember that for the Period, PeriodWithinYear, Polygon and WithoutPolygon rules one text file per species (taxa) is required.

Once you are satisfied that the rules are OK zip them into a folder and submit them to data@nbn.org.uk. The rules will then be loaded onto a NBN server and made available to all users of the NBN Record Cleaner. The same procedure is used to update the rules; the old zipped folder will be replaced with the new updated zipped folder on the NBN server so that update to the rules will be available for all users of the NBN Record Cleaner tool.

2.1. Guidance on the naming of rules

- The name of the zipped folder containing the rules should consist of your organisation abbreviation and rule type. Generally identification rules should be placed in a separate zip file. So for example Butterfly Conservation's National Moth Recording Scheme two zipped rule files for moth rules - **NMRS_Idifficulty.zip** (containing the Identification Difficulty rules) and **NMRS_Moth_rules.zip** (containing rules for range, recording and flight periods of moths).
- For larger taxonomic groups the range, recording and flight periods may be supplied as separate zipped files. The Botanical Society of the British Isles supplied two zipped folders - **BSBI_period_rules.zip** and **BSBI_range_rules.zip.**
- Rules grouped according to type will appear with in the NBN Record Cleaner listed under your organisation name.
- The Group row in the metadata section of each rule file should generally refer to the taxon group and rule type as appropriate. So for the moth example the rules are grouped into "**Moths flight Period**" (PeriodWithinYear rule), "**Moths Recording Period**" (Period rule) and "**Moths 10km distribution**" (WithoutPolygon rule). Rules not referring to species checks should be named so that the nature of the rule is apparent.
- For species rules the ShortName row in the metadata section of each rule file should generally refer to the species name to which the rule is applied. For these names consider whether the use of scientific names or common names is more appropriate, for example for birds would users prefer *Lullula arborea* or Wood Lark.

For further help with creating verification rules, please contact data@nbn.org.uk

3. Rule file format

3.1.Basic format

Rule files use INI format and are simple ASCII text. They can be edited with a text editor like Notepad. The basic structure is:

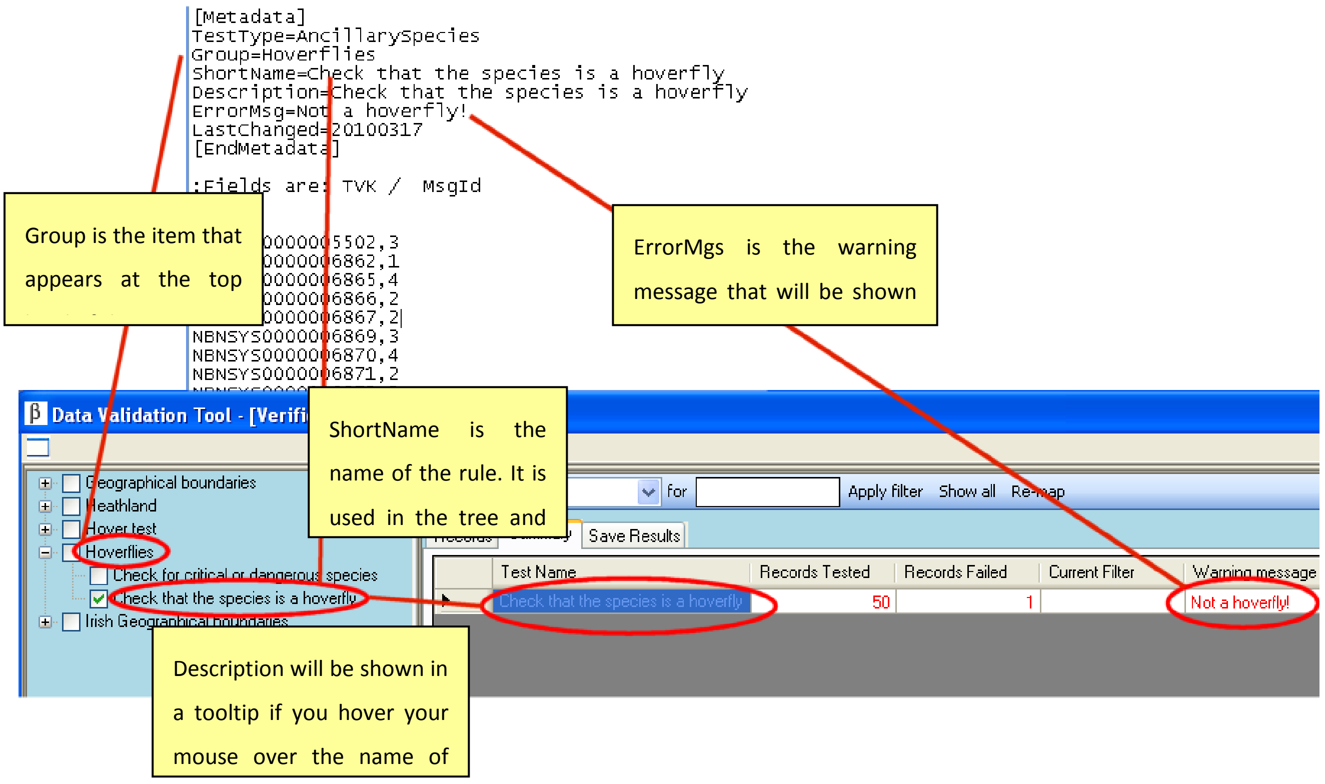
[Metadata]
TestType = *keyword specifying type of test*
Group = *the term that will appear at the top level of the rule tree*
ShortName= *the name of the rule in the rule tree*
Description= *longer description that appears as a tool tip*
ErrorMsg = *text that will be shown for failed rows*
LastChanged = *date the rule file was last updated*
… *other parameters - depending on the type of test*
[EndMetadata]

[Data]
the data against which rows will be tested – depending on the type of test

Dates are specified in “yyyymmdd” format: e.g. 15th March 2010 is specified as “20100315”.

Example:

This rule checks that records are of hoverflies. This is how it will look when displayed in the application:



```
[Metadata]
TestType=AncillarySpecies
Group=Hoverflies
ShortName=Check that the species is a hoverfly
Description=Check that the species is a hoverfly
ErrorMsg=Not a hoverfly!
LastChanged=20100317
[EndMetadata]

;Fields are: TVK /  MsgId

0000005502,3
0000006862,1
0000006865,4
0000006866,2
0000006867,2
NBNSYS0000006869,3
NBNSYS0000006870,4
NBNSYS0000006871,2
```

Group is the item that appears at the top

ErrorMgs is the warning message that will be shown

ShortName is the name of the rule. It is used in the tree and

Description will be shown in a tooltip if you hover your mouse over the name of

Test Name	Records Tested	Records Failed	Current Filter	Warning message
Check that the species is a hoverfly	50	1		Not a hoverfly!

Data Validation Tool - [Verifi
Geographical boundaries
Heathland
Hover test
Hoverflies
Check for critical or dangerous species
Check that the species is a hoverfly
Irish Geographical boundaries
for
Apply filter
Show all
Re-map
Save Results

3.2.Rule types

The following types of tests are supported.

TestType	Explanation
AncillarySpecies	Specifies a list of species that are expected to be found.
CheckList	This checks that the value of a field contains only a predetermined set of values.
Period	Checks that records are within a date range.
PeriodWithinYear	Allows records to be checked for being within a range of dates in any year (i.e. the record is for an appropriate season).
Polygon	This checks that the coordinates of the record are within the polygons supplied.
WithoutPolygon	Checks that grid squares or Vice-counties are in the list specified.
RegularExpression	Checks that the content of a field contains values that pass the specified regular expression check.
IdentificationDifficulty	Supply identification difficulty tags (in the range 1-5) for species.

3.2.1. Ancillary species list

A list of species that are expected to be found. A warning is generated if the species is not in the list.

```
[Metadata]
TestType=AncillarySpecies
Group=Heathland
ShortName=Checks species is expected in Heathland
Description=Checks if the species is in list of species that is accepted as typical for Heathland
ErrorMsg=Not a heathland species
LastChanged=20080911
[EndMetadata]

;Fields are: TVK

[Data]
BMSSYS0000001386
BMSSYS0000001612
```

Example of Ancillary Species rule file used to check that species recorded are typical heathland species

Note that

- There is no need to specify a field since the rule uses the species field.
- A comment can also be added to each species. These comments are included in the optional [INI] section linked to the species' taxonversionkey using MsgId column in the [Data] section.

```
[Metadata]
TestType=AncillarySpecies
Group=Heathland
ShortName=Checks species is expected in Heathland
Description=Checks if the species is in list of species that is accepted as typical for Heathland
ErrorMsg=Not a heathland species
LastChanged=20080911
[EndMetadata]

; define the comments associated with the species
[INI]
1=Rarely found on heathland.
2=This species rarely occurs in the UK so should be verified.
3=This species is protected.

;Fields are: TVK, MsgId
[Data]
BMSSYS0000001386,1
BMSSYS0000001612,2
```

Example of Ancillary Species rule file used to check that species recorded are typical heathland species, with the addition of comments for each species.

3.2.2. Check list

A list of values that are expected to be found. A warning is generated if the value is not in the list.

```
[Metadata]
TestType=CheckList
Group=Geographical boundaries
ShortName=Vice counties
Description=Check that the record has a valid vice county associated with it
DataFieldName=ViceCounty
ErrorMsg=Not a valid vice county
LastChanged=20092205
[EndMetadata]

[Data]
West Cornwall with Scilly,1
West Cornwall,1
East Cornwall,2
W. Cornwall,1
E. Cornwall,2
```

Example of the Check List rule file used to check values in ViceCounty field are valid vice county names.

Note

- Field to be checked is given in the DataFieldName field in the [Metadata] section.
- Values to be checked are given in the [Data] section.
- An optional second parameter on each Data row groups related records together which match this parameter (e.g. vice county number). In the example, either "West Cornwall with Scilly" or "West Cornwall" or "W. Cornwall" would map to the second parameter, 1 – so all are acceptable ways of specifying ViceCounty 1.

3.2.3. Period

Records checked for being within a date range. A warning is generated if the record is recorded outside this date range.

```
[Metadata]
TestType=Period
Group=Plant
ShortName=Amanita rubescens date range
Description= Check Amanita rubescens data is since introduction
Tvk=BMSSYS0000001386
ErrorMsg=Date outside range
StartDate=19540101
EndDate=20000101
LastChanged=20080911
[EndMetadata]
```

Example of the Period rule file used to check if records are recorded within the date range specified for *Amanita rubescens*

Note

- There is no need to specify fields because it has to use species and date fields.
- The test checks records with the specified TVK for dates within the date range specified.
- The file can be called anything *.txt – e.g. Amanita rubescensPeriod.txt
- One date can be left open to allow for records before or after a date

```
[Metadata]
TestType=Period
Group=Plant
ShortName=Impatiens glandulifera introduction
Description= Check Impatiens glandulifera data is since introduction
Tvk=NBNSYS9901234567
ErrorMsg=Date too early
StartDate=19540101
EndDate=
LastChanged=20080911
[EndMetadata]
```

Example of an open ended Period rule file used to check if records are recorded before the date specified for *Impatiens glandulifera*

3.2.4. Period within year

Records checked for being within a date range within a year. A warning is generated if the record is recorded outside this season date range.

```
[Metadata]
TestType=PeriodWithinYear
Group=Birds
ShortName=Adult out of range
Description=Check swallows seasonality is between May and August
Tvk=NBNSYS0000010326
ErrorMsg=Adult out of range
LastChanged=20080911
StartDate=0601
EndDate=0908
[EndMetadata]
```

Example of a PeriodWithinYear rule file used to check if swallow records are recorded outside their appropriate season

Note

- There is no need to specify fields because it has to use species and date fields.
- The test checks records with the specified TVK for dates within the date range specified.
- Dates are in MMDD format
- The file can be called anything *.txt – e.g. SwallowSeason.txt
- A stage can be optionally specified as a comma delimited list of alternative text descriptions

```
[Metadata]
TestType=PeriodWithinYear
Group=Birds
ShortName=Adult out of range
Description=Check swallows seasonality is between May and August
Tvk=NBNSYS0000010326
ErrorMsg=Adult out of range
LastChanged=20080911
DataFieldName=SpeciesStage
StartDate=0601
EndDate=0908
[EndMetadata]

[Data]
Stage=A,Adult
StartDate=0501
EndDate=0608
```

Example of a PeriodWithinYear rule file used to check if adult swallow records are recorded outside their appropriate season

- If one (or more) sections are listed in the Data section then DataFieldName is required and the test will only be offered if the data has a field of that name (e.g. SpeciesStage in example below) and if at least some of the records contain the appropriate TVK.
- Stage in data section can be comma delimited list of term options (i.e. Ad, juv etc)
- Both the date in the Metadata and in the Data section are tested if supplied.

3.2.5. Polygons

The coordinates of the records are within the polygons supplied. A warning is generated if the record is recorded outside the polygon.

```
[Metadata]
TestType=Polygon
Group=Hoverfly range
ShortName=Baccha elongata
Description=Checks range is within know range of Baccha elongata
EPSG=27700
Shapefile=BacElo.shp
ShapeRecordId=
ShapeColPos=
DataRecordId=NBNSYS0000006862
DataFieldName=Species
ErrorMsg=Record outside known range of Baccha elongata
LastChanged=20090612
[EndMetadata]
```

Example of a Polygon rule file used to check against known range of *Baccha elongata*

Note

- This rule uses polygon(s) from which it builds an index of grid squares as a one off task, if this index list is not pre-supplied.
- DataFieldName is the name of field that contains data to be filtered (e.g. Species (for TVK, speciesname etc). If the DataFieldName is Species and a TVK is supplied, the test will only be offered if there are records of the specified species. If this field is left empty then its applied to all records
- DataFieldRecordId: For a species the TVK (which will be translated into a PTVK). For anything else the value of the field as supplied.
- Shapefile can either have full path or no path. If no path, it is assumed to be in the same folder as the .txt file. You must provide a shape file in EPSG 27700 (British E/N), even if the area covered is Ireland.

- ShapeRecordId= (i.e. left blank) then no filtering is used and all polygons in the shapefile are used.
- ShapeColPos is a zero based array pointing to the column in the dbf file that contains the ShapeRecordId (if used)
- EPSG must match the coordinate system used in the shapefile. It should always be 27700
- [GridCoarse] and [GridDetailed] are calculated automatically from the Polygon (and saved into the file) the first time the rule is used if these are found to be missing. This means that users need write access to the configuration file. This can be a time consuming operation, so it is preferable to provide the file with the index already populated. The index information can also be supplied without providing the shape file in which case only approximate checks can be undertaken using 10km and 1km squares. The second parameter (after the #) indicates completely within (1) or partially within (0)

```
[Metadata]
TestType=Polygon
Group=Geographical boundaries
ShortName=VC 1
Description=Checks location is within vice county 1
Field=
EPSG=27700
Shapefile=vc.shp
ShapeRecordId=1
ShapeColPos=0
DataRecordId=
DataFieldName=
ErrorMsg=Coordinate not in vice county 1
LastChanged=20090531
[EndMetadata]

#Grid square followed by 1 if it is completely within or 0 if it's a partial overlap
[GridCoarse]
8,0#0
8,1#0
9,0#0
9,1#0

[GridDetailed]
503,193#0
504,168#0
504,170#1
504,171#1
504,172#1
```

Example of a Polygon rule file used to check records against known vice county. Index ([GridCoarse] and [GridDetailed]) pre-supplied.

3.2.6. Without Polygon Check

The coordinates of the records are within grid squares or vice-counties listed. A warning is generated if the record's grid square or vice-county is outside this list.

```
[Metadata]
TestType=WithoutPolygon
Group=Species Range
ShortName=Impatiens glandulifera 1km
Description=Check coordinate against known distribution of Impatiens glandulifera
ErrorMsg=Coordinate outside known range for Impatiens glandulifera
DataFieldName=Species
DataRecordId=NBNSYS0000002278
LastChanged=20090527
[EndMetadata]
[VC_GB]
1
2
3

[VC_Ireland]
1
2
3

OR
[10km_GB]
SM12
SM13

[10km_Ireland]
H11

[10km_CI]
```

Example of a WithoutPolygon rule file used to check records of *Impatiens glanduilfera* against vice county or 10km square lists.

For 5km, 2km or 1km squares replace 10km with appropriate square resolution. For 5km and 2km squares use appropriate letter at end of grid reference to denote the square within the corresponding 10km square eg 5km squares SM12SE, SM12NE, SM12NW, SM12SW, 2km squares (DINTY system) SM12D, SM12F, SM12Z etc.

Great Britain 2km square example

```
[2km_GB]
SM12A
SP24B
TL24E
NS98X
```

Note

- It is possible to list EITHER vice counties OR 10km OR 5km OR 2km OR 1km squares that the eg. species occurs in.
- Rule can be called anything *.txt – e.g. ImpatiendGlanduliferaRange.txt
- DataFieldName is the name of field that contains data to be filtered (e.g. Species, or ViceCounty – which will get mapped to appropriate user supplied field)
- DataFieldRecordId: For a vice county the number, for a species the TVK (which will be translated into a PTVK). For anything else the value of the field as supplied.
- VC_GB or VC_Ireland is a list of numeric codes for the Vice-counties in which the species has been found.
- _GB and _Ireland the system will only perform this test on records that have their coordinates in the correct system (e.g. if only 1km_GB were provided the test would be ignored for any Irish records).

3.2.7. Regular expressions

Values in a field checked against a regular expression. A warning is generated if the value does not pass this regular expression check.

```
[Metadata]
TestType=RegularExpression
Group=Other checks
ShortName=Checks postcode is valid
Description=Checks that the record has a valid format postcode entered
DataFieldName=Postcode
ErrorMsg=Not a valid post code
LastChanged=20080911
[EndMetadata]

[Expression]
^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$
```

Example of a Regular Expression rule file used to check for valid postcodes

Note

- Field to be checked is given in the DataFieldName field in the [Metadata] section

Regular expressions are very powerful and can be quite complex. There are some good web-sites providing tutorials and examples. Googling “regular expressions” should provide plenty of information.

3.3. Rule reversal

Any rule type can be reversed by including "ReverseRule=True" in the [Metadata] section. A warning is generated if the record passes the rule check.

The two most likely scenarios are the use of the reverse rule is for an AncillarySpecies list to detect species which ARE in the given list (rather than are not in the supplied list) and use of the Polygon or WithoutPolygon to detect records which fall WITHIN the specified area (rather than outside it). This could be used, for example, to check for marine species occurring within a coastal outline (i.e. on land).

```
Metadata]
TestType=AncillarySpecies
Group=Alert list
ShortName=Check for species we need to know about.
Description=Checks for records of species where some follow-up action is necessary.
ErrorMsg=Alert species!
ReverseRule=True
LastChanged=20101119
[EndMetadata]

 [Data]
BMSSYS0000001386
BMSSYS0000001612
NBNSYS0000005502
NBNSYS0000006862
NBNSYS0000006865
NBNSYS0000006866
NBNSYS0000006867
NBNSYS0000006869
```

Example of reversal of AncillarySpecies rule to alert for records of specific species

3.4. Identification difficulty rules

Identification difficulty of species records can be highlighted in the verification process

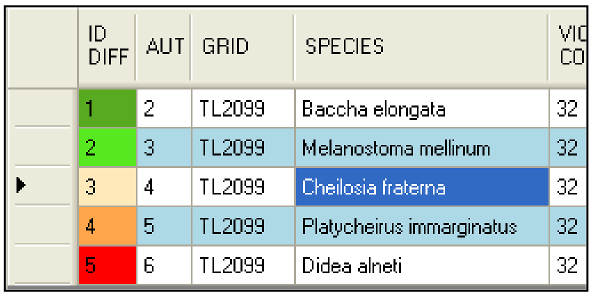
These follow the same basic format as a rule file although their purpose is slightly different. They do not apply a pass/fail type test, but supply tags for species that are used in the result grid of the Verification screen to alert users to species where there may be identification or nomenclatural problems.

```
[Metadata]
TestType=IdentificationDifficulty
Organisation=Hoverfly Recording Scheme
LastChanged=20101019
[EndMetadata]

; Define the messages allocated to the tags. 1-5 are available.
; It is not necessary to use them all!
[INI]
1=Can be identified at sight in the field.
2=Can be identified in the field with care and experience.
3=Needs confirmation from vice county recorder.
4=Needs confirmation from national expert.
5=Voucher specimen required to be examined by national expert.

; TVK = MsgID
[Data]
NBNSYS0000005502=3
NBNSYS0000006862=1
NBNSYS0000006865=4
NBNSYS0000006866=2
NBNSYS0000006867=2
NBNSYS0000006869=3
```

Example of IdentificationDifficulty rule for hoverfly species

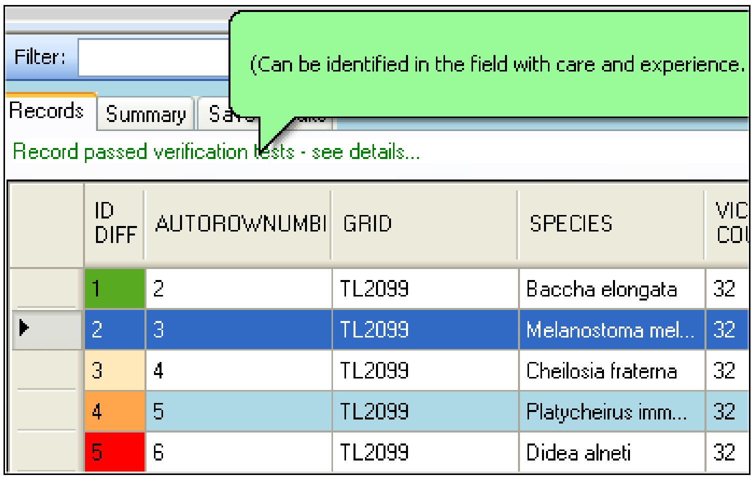
Note

- Tags 1-5 are available (but you don't have to use them all).
- Tag 1 should indicate that the species is easy to identify, not requiring further determination.
- The following screen shot shows the colour coding used in the Verification screen grid in the application:



	ID DIFF	AUT	GRID	SPECIES	VIC CO
	1	2	TL2099	Baccha elongata	32
	2	3	TL2099	Melanostoma mellinum	32
▶	3	4	TL2099	Cheilosia fraterna	32
	4	5	TL2099	Platycheirus immarginatus	32
	5	6	TL2099	Didea alneti	32

The messages attached to the tags will be shown in the usual tool-tip.



(Can be identified in the field with care and experience.

Record passed verification tests - see details...

	ID DIFF	AUTOROWNUMBI	GRID	SPECIES	VIC COI
	1	2	TL2099	Baccha elongata	32
▶	2	3	TL2099	Melanostoma mel...	32
	3	4	TL2099	Cheilosia fraterna	32
	4	5	TL2099	Platycheirus imm...	32
	5	6	TL2099	Didea alneti	32

4. External script files

These allow custom external modules to be automatically imported into the system and offered as tests. For the vast majority of uses, it is not anticipated that external script modules will be needed. The flexibility of the standard test types should be sufficient. Scripts are supported for the rare occasions when an additional test is required that is not achievable in any other way. This is most likely to be the case if a test requires data from a combination of two or more columns. The standard test types mostly apply tests to data in a single column. (The exception is period tests which can use data supplied by an additional Stage column to apply different date ranges for different life stages.)

An example of a situation where a script file could be useful arises because some species can only be reliably identified using characters of the male genitalia. You might wish to define a test which fails records of such species unless the sex is “male”. This requires a test which uses a combination of the species identity and the sex (which would need to come from an additional data column).

External script files are simply text files containing un-compiled VB.NET code, but must have a .srcx extension and be located in the same folder as the other tests. The application will load external script modules along with any other rule files it finds and get information from them about the field(s) and/or taxa they work with so that the system knows whether to offer the checks they include. So, in our example of the male record check, such a test would only be offered if the imported data did include a “sex” column and also included records of the species to which it applies.

In addition:

- They inherit all imported references from the main application.
- Classes and methods that exist in the main application are available for use in script modules.
- Additional functions or sub routines can be added in class module.

- Additional classes can even be added into the text file.

Fields from the imported data that is being tested that can be referenced from scripts are:

Field name	**Description**
PTVKFIELD	NBN preferred taxon version key
TVKFIELD	NBN taxon version key
SPECIESFIELD	Species name
SPECIESID	How hard the species is to identify (1-5)
ErrorDesc	Description of any errors
Precision	Precision of coordinate
EPSG	EPSG of coordinate (normally 27700)
Transformed_X	Coordinate in British grid (epsg: 27700)
Transformed_Y	Coordinate in British grid (epsg: 27700)
CoordKey	Concatenated string giving unique grid square on map e.g. 328000,260000,1000,27700 Transformed_X,Transformed_Y,Precision,epsg
DateFrom	Date from (also used if only single date provided)
DateTo	Date to
VagueDateDisplay	Date displayed as vague date
RECORDKEYFIELD	Unique record id
SAMPLEDATEFROMFIELD	Validated Date From
SAMPLEDATETOFIELD	Validated Date To (may not be present if on single date provided)
COORDXFIELD	Coordinate provided by the user – may be X or GridRef
COORDYFIELD	Y coordinate provided by user (may not be present if user provided grid ref)
CountyField	Vice county field – as number

If you have imported additional data columns they will also be available. They will have the same name as they had in the original data file.

Two examples of external scripts are given below

- TestScript1.srcx is a trivial example which tests whether the species name is "HYACINTHOIDES NON-SCRIPTA". It will pass records where this is true and fail any others. This test should always appear in the tree of available rules since it only depends on a species name field being available which should always be the case.

- TestScript2.srcx contains a very simple implementation of the male sex test described above. It checks for record of species of the hoverfly genus *Sphaerophoria* and, if the data has a "sex" column, checks that this contains a term starting with "M". This test will only be offered if the imported data contains a Sex column and includes records of one or more of the specified species.

4.1.TestScript1

Here is the code. The bits you would need to change to implement a simple test of your own are shown in red:

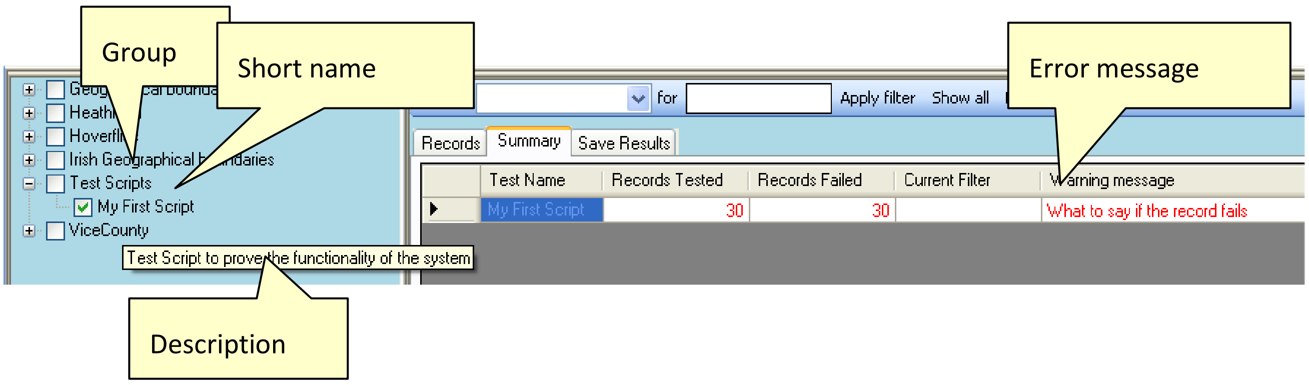
```
Imports ScriptedTestsInterface

Public Class DynamicCode

#Region " Class Interfaces "
    Implements iScriptedTests
#End Region

#Region " Class Scoped Objects "
    Private mValidationTestResults As ValidationTestResults
    Private mTestKey As String
#End Region


#Region " Class Event Prototype "
    Public Event Progress(ByVal PercentComplete As Integer, ByVal Message As String) Implements
      ScriptedTestsInterface.iScriptedTests.Progress
#End Region


#Region " Mandatory Interface Properties "

    Public ReadOnly Property Description() As String Implements
      ScriptedTestsInterface.iScriptedTests.Description
        Get
            'Add a description of your test that is displayed to users
            Return "Test Script to prove the functionality of the system"
        End Get
    End Property

    Public ReadOnly Property ErrorMessage() As String Implements
      ScriptedTestsInterface.iScriptedTests.ErrorMessage
        Get
            'Give a message if a record fails a test
            Return "What to say if the record fails"
        End Get
    End Property

    Public ReadOnly Property Group() As String Implements ScriptedTestsInterface.iScriptedTests.Group
        Get
            'Control which group in the tree the test should appear in
            Return "Test Scripts"
        End Get
    End Property

    Public ReadOnly Property ShortName() As String Implements
      ScriptedTestsInterface.iScriptedTests.ShortName
        Get
            'Give your test a name
            Return "My First Script"
        End Get
    End Property

    Public Property Key() As String Implements ScriptedTestsInterface.iScriptedTests.Key
        Get
            Return mTestKey
        End Get
        Set(ByVal value As String)
            mTestKey = value
        End Set
    End Property


    Public Property Results() As ValidationTestResults Implements
      ScriptedTestsInterface.iScriptedTests.Results
        'Structure to return test results in - should not need editing
        Get
            Return mValidationTestResults
        End Get
        Set(ByVal value As ValidationTestResults)
            mValidationTestResults = value
        End Set
    End Property

    Public ReadOnly Property TestField() As String Implements
      ScriptedTestsInterface.iScriptedTests.TestField
        Get
            'Specify which field your test acts on - the system will check that this field exists in
      your data before making the test available
            Return "SpeciesField"
        End Get
    End Property


    Public ReadOnly Property TVK() As String Implements ScriptedTestsInterface.iScriptedTests.TVK
        Get
            'Optional
            'Specify which species your test acts on - the system will check that these species exist
      in your data before making the test available
            'Can be , delimited list for more than one tvk
            Return ""
        End Get
            End Property

#End Region


#Region " Method to Run tests "

    Public Function RunScript(ByRef dv As System.Data.DataView) As Boolean Implements
      ScriptedTestsInterface.iScriptedTests.RunScript
        RunScript = True

        Try

          'Provide % complete for status bar
            Dim intCounter As Integer = 0
            Dim intDivider As Integer = CInt(dv.Count / 100) + 1

            Dim blnOK As Boolean = False
            Dim r As DataRowView

            Dim NewValidationTestResults As New ValidationTestResults
            NewValidationTestResults.TotalRecordsTested = dv.Count
            NewValidationTestResults.ActiveRowFilter = dv.RowFilter
            NewValidationTestResults.FailedCount = 0

            For Each r In dv

                intCounter += 1
                If (CInt(intCounter Mod intDivider) = 0) Then
                    RaiseEvent Progress(CInt(intCounter / intDivider), Me.ShortName)
                End If


        ' Your code start
                '========================================================================
                '
                ' your code goes in here Setting blnOK to false in the event of a failure
                '
                ' This example is trivial and returns and error if the species
                ' is not Hyacinthoides non-scripta

                  blnOK = r.Item("SpeciesField").ToString.ToUpper = "HYACINTHOIDES NON-SCRIPTA"

                '========================================================================
                'Your code ends

```

```
140         'Build test results - should not need editing
141                 If Not blnOK Then
142                     NewValidationTestResults.FailedCount += 1
143                     If Not r("ErrorDesc").ToString.Contains("|" & mTestKey.Trim & "|") Then
144                         r("ErrorDesc") = r("ErrorDesc").ToString & "|" & mTestKey.Trim & "|"
145                     End If
146                 End If
147 
148             Next
149 
150             mValidationTestResults = NewValidationTestResults
151 
152 
153         Catch ex As Exception
154             MsgBox(ex.Message)
155             RunScript = False
156         End Try
157 
158         Return RunScript
159 
160     End Function
161 
162 #End Region
163 
164 
165 End Class
```

You would need to edit the properties that are returned in lines 23-52 to implement your own test.

Line	Property	Value returned in example	Explanation
27	Description	Test Script to prove the functionality of the system	The tool tip that will be shown if you hover your mouse over the name of the test in the tree of rules
35	Error Message	What to say if the record fails	The warning message that is displayed if the test fails
39	Group	Test Scripts	The top level item in the tree of rules under which this test will be listed
50	ShortName	My First Script	The name of this test in the tree of rules

This is how it looks in the application:



TestField() in Lines 75-82 specify the field, or fields, to test. The value returned in the example in line 80, “SpeciesField” is the field containing species names (see table above). The string returned can be a comma delimited list of field names if the test relies on two or more columns being present. The test will only be offered if the fields named here are available in the imported data. Since the species field is required, this particular field will always be available. Therefore, this particular test will always be offered.

Line 134 contains the actual test:

```
blnOK = r.Item("SpeciesField").ToString.ToUpper = "HYACINTHOIDES NON-SCRIPTA"
```

This obtains the item from the SpeciesField, forces it to upper case, and tests it against the string "HYACINTHOIDES NON-SCRIPTA". The result is stored in the variable blnOK which is used to return the result of the test to the application. It will be TRUE if the name matches this string, or FALSE otherwise.

We only needed to change just six lines – the four properties, the field name that will be tested and the actual test itself (which may of course require a bit more code to do something useful!) in TestScript1 to implement a simple rule!

4.2.TestScript2

The purpose of this test is to check that records of *Sphaerophoria* are based on males, since these species can only be identified reliably using characters of the male genitalia.

As before, we need to set the properties for our test:

```
    Public ReadOnly Property Description() As String Implements
      ScriptedTestsInterface.iScriptedTests.Description
        Get
            'Add a description of your test that is displayed to users
            Return " Test that records of Sphaerophoria species are based on males"
        End Get
    End Property

    Public ReadOnly Property ErrorMessage() As String Implements
      ScriptedTestsInterface.iScriptedTests.ErrorMessage
        Get
            'Give a message if a record fails a test
            Return " Identification only possible for males."
        End Get
    End Property

    Public ReadOnly Property Group() As String Implements ScriptedTestsInterface.iScriptedTests.Group
        Get
            'Control which group in the tree the test should appear in
            Return "Test Scripts"
        End Get
    End Property

    Public ReadOnly Property ShortName() As String Implements
      ScriptedTestsInterface.iScriptedTests.ShortName
        Get
            'Give your test a name
            Return " Sphaerophoria male"
        End Get
    End Property
```

We will detect the species by checking the PTVK and we will attempt to get the sex from an additional data column called “Sex”. Of course, we cannot do the test if the Sex column is not available. Therefore, we want to test for the presence of the PTVKFIELD and an additional SEX column. (Actually, the test for PTVKFIELD is probably unnecessary because we know that is always going to be there, but it provides an example of testing for multiple columns.)

```
    Public ReadOnly Property TestField() As String Implements
      ScriptedTestsInterface.iScriptedTests.TestField
        Get
            'Specify which field your test acts on - the system will check that this field exists in
      your data before making the test available
            Return "PTVKFIELD,Sex"
        End Get
    End Property
```

We only want the test to be offered if there are records of the species we are interested in. We need to know the PTVKs of these species. They can be found from MasterSpeciesList.txt (or the NBN dictionary).

Species	PTVK
Sphaerophoria bankowskae	NBNSYS0000033254
Sphaerophoria batava	NBNSYS0000006954
Sphaerophoria fatarum	NBNSYS0000006953
Sphaerophoria interrupta	NBNSYS0000157028
Sphaerophoria loewi	NBNSYS0000006955
Sphaerophoria philanthus	NBNSYS0000157031
Sphaerophoria potentillae	NBNSYS0000006963
Sphaerophoria rueppellii	NBNSYS0000006958
Sphaerophoria scripta	NBNSYS0000006959
Sphaerophoria taeniata	NBNSYS0000006960
Sphaerophoria virgata	NBNSYS0000006961

We can pass the list of TVKs we are interested in to the TVK property:

```
    Public ReadOnly Property TVK() As String Implements ScriptedTestsInterface.iScriptedTests.TVK
        Get
            'Optional
            'Specify which species your test acts on - the system will check that these species exist
      in your data before making the test available
            'Can be , delimited list for more than one tvk
            Return "NBNSYS0000033254,NBNSYS0000006954,NBNSYS0000006953,… ,NBNSYS0000006961"
        End Get
            End Property
```

Note that the list has been abbreviated here so that the line fits on the page – the full list appears in the actual file.

There is a bit more code involved in the test this time:

```
        Try

          'Provide % complete for status bar
            Dim intCounter As Integer = 0
            Dim intDivider As Integer = CInt(dv.Count / 100) + 1

            Dim blnOK As Boolean = False
            Dim r As DataRowView

            Dim tvk as string
            Dim tvkList as string = "NBNSYS0000033254,NBNSYS0000006954,… ,NBNSYS0000006961"
            Dim sx as string

            Dim NewValidationTestResults As New ValidationTestResults
            NewValidationTestResults.TotalRecordsTested = dv.Count
            NewValidationTestResults.ActiveRowFilter = dv.RowFilter
            NewValidationTestResults.FailedCount = 0

            For Each r In dv

                intCounter += 1
                If (CInt(intCounter Mod intDivider) = 0) Then
                    RaiseEvent Progress(CInt(intCounter / intDivider), Me.ShortName)
                End If


                 ' Your code start
                '========================================================================
                '
                ' your code goes in here Setting blnOK to false in the event of a failure
                '
                blnOK = True
                tvk = r.Item("PTVKFIELD").ToString
                If InStr(tvkList, tvk) > 0 then
                    sx = r.Item("SEX").ToString.ToUpper
                    If Len(sx) > 0 then
                        blnOK = Left(sx, 1) = "M"
                    End If
                End If
                '========================================================================
                'Your code ends


        'Build test results - should not need editing
                If Not blnOK Then
                    NewValidationTestResults.FailedCount += 1
                    If Not r("ErrorDesc").ToString.Contains("|" & mTestKey.Trim & "|") Then
                        r("ErrorDesc") = r("ErrorDesc").ToString & "|" & mTestKey.Trim & "|"
                    End If
                End If

            Next

            mValidationTestResults = NewValidationTestResults


        Catch ex As Exception
            MsgBox(ex.Message)
            RunScript = False
        End Try
```

First we declare string variables “tvk” to hold the PTVK, “tvkList” which is initialised to the list of TVKs (abbreviated in the code sample shown) and “sx” to hold the value from the Sex column.

In the “Your code start” to “Your code end” block, we start by setting blnOK to True. If we cannot complete the test (because the record is not of one of the species we are interested in or because we cannot get a value for the sex) we don’t want it to fail.

Next we retrieve the PTVK from PTVKFIELD and test whether it is in the list of TVKs we are interested in. If it is, we attempt to retrieve the sex and force it into upper case.

Finally, if we got a value for sex, we check whether it starts with “M”. If it doesn’t then we have record of something other than a male and the test fails.

This code could easily be improved. An obvious thing to do would be to read from a file the list of TVKs, and perhaps also the values of Sex which are accepted or rejected. Users could then easily configure the species they wanted checked in this way without needing to mess with the script file. I leave that to the reader as homework!

4.3.Hint for those developing scripts

Once you have got to the verification screen with an example dataset, if your test fails or doesn’t initially display, you should normally be able to edit your test (in Notepad or whatever text editor you prefer) and then simply use the [Options] button underneath the tree to reload the test. This saves you having to rerun the application and then load and check your test data from scratch each time you make a change.