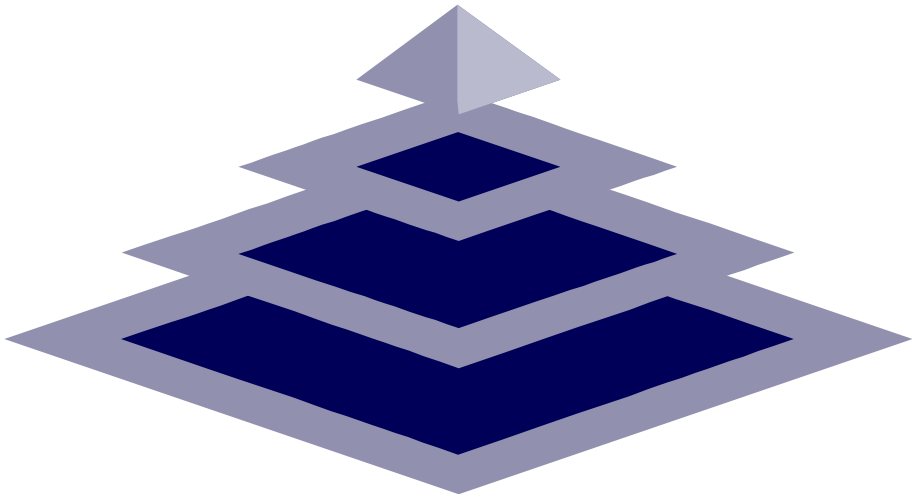
DORSET SOFTWARE

RECORDER



APPLICATION EXTENSIBILITY SPECIFICATION

FOR JNCC

Issue History

VERSION	APPLIES TO APPLICATION VERSION	DATE
1	1.0.0.0	21/12/1999
2	1.0.1.0	31/01/2000
3	1.0.2.0	29/02/2000
4	1.0.3.0	31/03/2000
5	1.0.5.0	12/06/2000
6	1.0.6.0	07/07/2000
7	1.0.8.0	27/02/2001
8	1.1.0.0	23/04/2001
9	1.1.0.1	22/05/2001
10	1.1.1.0	03/08/2001
11	1.1.2.4	13/12/2001
12	1.1.3.0	09/01/2002
13	3.0.0	21/01/2003
14	6.1.0	20/02/2003
15	6.2.0	13/03/2003
16	6.2.0	02/04/2003
17	6.2.3	24/05/2004
18	6.5.1	30/07/2004
19	6.6.1	15/12/2004
20	6.10.1	16/02/2007

Document Change Log

DATE	CHAPTER	DESCRIPTION OF CHANGE	NAME
08/01/2002	IRecorder Functions	Added ISpatialRefSystem	JvB
15/01/2003	IRecorder2000	Added ConvertAccessSQLToSQLServerSQL, ConnectionString and SetApplicationSecurity to IRecorder2000.	JvB
20/02/2003	IRecorder2000, IRecorderFunctions	Changes to CheckDatabase and MergeDatabase, added GetFilterItems and ExportKeyList for CCN90.	JvB
07/03/2003	IReportResults	Added new interface specification.	JvB
11/03/2003	IProvidesOccurrencesSQL	Added new interface specification.	JvB
31/03/2003	Creating COM objects in Delphi 4	Updated DMAP addin example to inherit from TAutoObject.	JvB
22/05/2003	IBaseMapFormat	Added missing functions LatLongToMapCoordX and LatLongToMapCoordY to IBaseMapFormat interface.	JvB
06/01/2004	General	Added several new interface specifications for multiple map support, occurrence addin support. Added IRecorder2000. RequestCOMData and extensions for menu management and Tools..Options dialog pages.	JvB
26/02/2004	IAvailableMap	Added IMapDropFormat, IAvailableMap, INodeManager interfaces.	ES
24/06/2004	IRecorder6	Added IRecorder6, with DisplayTab and DisplayDataFromSQL methods	JvB
26/07/2004	IRecorder2000, IRecorder6	Added 'Feature' datatype to DisplayTab, DisplayDataFromSQL and DisplayData methods.	JvB
11/08/2004	IRecorder2000	RequestCOMData now a function that returns the addin instance used to obtain data.	JvB

02/09/2004	Hierarchy Nodes Interfaces	Updated the description of the properties and functions.	ES
15/12/2004	IRecorder6	Added a note of the special case for DisplayTab when the recurse parameter is set but the expand is NOT.	KJG
09/12/2005	ICurrentSettings6	Added SessionID property.	JvB
11/08/2006	IGridSystem, IENBox	Added interface for support of grid square spatial reference systems	JvB
11/09/2006	IAdditionalPage6	Added details of IAdditionalPage6.	JvB
29/09/2006	IHelp	Clarification of functionality for IAdditionalPage classes that implement IHelp	JvB
23/01/2006	ISpatialReferenceList, IBaseMapFormatList, INamedSpatialSystem, IBaseMapScaleList	Added new interface descriptions	JvB
22/10/2007	IXmlReportList, IXmlReport, IKeyTypeList, IKeyType, IRecorder 6	New interfaces and methods to support Xml Reports from within addins.	JvB
07/01/2008	IValidation6	CCN232 – validation interfaces for validation of main database	JvB
21/03/2008	IExportFilter 6	CCN233 - Revalidation before export.	ES
14/10/2008	IBaseMapScaleList_614	Extension to IBaseMapScaleList to allow for grid offsetting.	ES

CONTENTS

Formatted: French (France)
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Formatted: French (France)

INTRODUCTION

This document details the full range of requirements for application extensibility in Recorder 2000 and, what and how interfaces need to be implemented to achieve these requirements. The chosen technology for achieving this is the Component Object Model (COM). COM uses interfaces which define properties, methods and events to achieve the required functionality. These interfaces are defined in a type library (see section below) and this type library can be accessed by both Recorder 2000 and add-in modules alike to provide a method of communication between them.

GLOSSARY

Term	Meaning
Active Form	Borland Delphi terminology for a specialised form type which when built provides an ActiveX component.
ActiveX Control	Microsoft's current technology for component reuse. ActiveX controls can be embedded in DLL or OCX files, and as they are based on COM technology they are language independent (though Windows dependent).
Automation object	A type of COM object, which supports OLE Automation. This allows it to declare its interfaces (i.e. what it is capable of doing) dynamically at run time.
Class	A collection of related properties, methods and events.
COM	Component Object Model, Microsoft's model for object sharing across processes in Windows.
COM object	An object capable of being used with Microsoft's COM technology.
Dispinterface	Dispatch interface types define the methods and properties that an Automation object implements through IDispatch. Required for defining Events that the client application can respond to via an event sink.
DLL	Dynamic Link Library. A library of code which can be linked to an executable at run-time, and can contain functions, and COM objects.
EXE	Executable application file.
GUID	Globally Unique Identifier. A GUID is a 16-byte binary value that uniquely identifies an

Formatted: Table Row

	interface, a class etc.
IDispatch	Along with IUnknown, an interface that all COM objects support that handles the invoking of methods through it's Invoke method.
Interface	A class can be divided into its data, its code, and the declaration of its behaviour to the outside world. This declaration is called an interface, which in COM technology can be defined in isolation from any class which may implement that interface.
IUnknown	Along with IDispatch, an interface that all COM objects support that handles the destruction of the COM object via reference counting (methods _AddRef and _Release handle this). Contains a third method, QueryInterface, which allows COM objects to be "questioned" whether they support an interface or not.
OCX	File extension used for ActiveX controls and libraries of ActiveX controls.
Server	A collection of code items which provides services to a client. For example, an OCX provides one or more ActiveX controls to the client (the main application).
Type Library	Collection of interfaces and their definitions.
VCL	Visual Component Library. The library of code, classes and components which is supplied with Delphi.

Formatted: Table Row

REQUIREMENTS

REQUIREMENT	DESCRIPTION
Validation	Allows a COM object to apply validation rules to data entering the system, either via NBN data import or via a data entry screen.
Replacement Form	Allows COM objects to replace any of the existing data entry forms.
Additional Page	Allows COM objects to add themselves to existing screens as an additional detail page for data entry screens with Page Controls. Because of locking considerations, additional pages cannot access the standard datasets for the current record. Data can be obtained from additional 'addin' tables, or by reading information from existing controls using the IRecorderForm interface.
Term Lists	New term lists are added to the data model directly and must adhere to relational database rules. See TSD for the format of term lists and how to make new term lists be accessed and edited via the term lists screen.
New Menu Items	Allows the specification of a new menu item (and toolbar button) which calls a form within the COM object. This could be a new map form (using GIS), a new wizard or any other form type.
Import/Export Filters	Allows new data import and export filters to be defined within the COM object.
Map filters	Allows map data to be imported and exported in a new format.
Spatial References	Allows new spatial referencing systems to be used from within the system. Provides a means for conversion to and from Lat/Long co-ordinate systems.
Record Card headers	Allows the specification of new headers for the Species for a Place record card.
Additional Filters dialog	Allows a new additional filter dialog to be connected to the Report Wizard.
Data Transfer	Recorder 2000 must be able to transfer data to and from COM objects. This is achieved by COM objects supporting the CF_JNCCDATA clipboard format as defined in the TSD which handles draggind and dropping, cutting and pasting data.

Current Settings	Recorder 2000 must expose the current rucksack, current survey, current taxon checklist, biotope classification and admin area type to COM objects.
Open Windows and Record Keys	Recorder 2000 must expose its open windows and key of the records they are currently displaying (where applicable) to COM objects.
Active Window Mode	Recorder 2000 must expose the current mode (Add, Edit, Browse) of its active window to COM objects.
Result Window Query	Recorder 2000 must expose the results in frmResultWindow to COM objects.

INTERFACES

By implementing interfaces within COM classes, it is possible to provide standard functionality to Recorder 2000 which allows the application to use the class for various different purposes. Classes implement one or more interfaces, and can combine interfaces to declare to Recorder 2000 which capabilities they posses. For example, a Replacement form class may also implement interfaces specifying file formats which that form can handle.

Interfaces implemented by Recorder 2000

The following set of interfaces are implemented within Recorder 2000 and provide access to various aspects of its data and functionality to external Addins.

IRecorder2000

This interface allows addins to interrogate Recorder 2000 settings and Rucksack contents via COM.

IRecorder2000 is implemented within Recorder 2000 by OLE Automation. Addins can create an instance of IRecorder2000 at any time, for example using the Delphi function CreateOLEObject. The OLE Object name is 'Recorder2000.AutoApplicationSettings'

Attribute	Type	Data Type	Description
CurrentSettings	Read only property	ICurrentSettings6	Supplies an interface pointer to the ICurrentSettings6 object to addins.
Rucksack	Read only property	IRucksack	Supplies an interface pointer to the IRucksack object to all addins.
JNCCFormat	Read only property	Integer	Supplies a constant to addins which represents the internal clipboard and drag/drop format.
MenuOptionClick	Procedure	None	Takes a Widestring parameter. This is a semi-colon separated list of the menu option and its hierarchy which is to be clicked. This allows addins to click any menu option available on the Recorder 2000 menu. For example, passing 'Tools;Options…' as a parameter displays the options screen.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

DisplayData	Procedure	None	Allows addins to display a screen filtered to a specified key list of data, in a fashion similar to related data. Takes 2 parameters, firstly, the type of data to display (widestring). Possible values are 'Survey', 'Event', 'Sample', 'Occurrence', 'Location', 'Feature', 'Name', 'Location', 'Document', 'Taxon', 'Biotope', 'Admin'. Alternative values are provided by addins which implement the IDisplayData interface (note for this to work, the COM form must be made active first). The second parameter is an IKeyList which specifies the data items to filter for. This should normally be preceded by a call to display the appropriate form. When the table to be called is of type 'Name', or an Occurrence, it is necessary to supply a mixed key list. In this case, the KeyField2 of the item in the key list must specify the table name (INDIVIDUAL or ORGANISATION for names, TAXON_OCCURRENCE or BIOTOPE_OCCURRENCE for occurrences).
RequestData	Procedure	None	Allows COM Addins to request data selection from other forms in a manner similar to Return Data functionality. Two parameters are required. The first is an implementation of the IRequestor interface. This provides a method which is called when data is returned. The second parameter is a widestring indication of the type of data which is to be returned. Possible values are Possible values are 'Survey', 'Event', 'Sample', 'Occurrence', 'Location', 'Name', 'Location', 'Document', 'Taxon', 'Biotope', 'Admin, 'MapPoint', 'MapBox'. Links cannot be setup unless the link is being requested from the currently active MDI form. When Name data is requested, the KeyField2 element of each item indicates INDIVUDUAL or ORGANISATION and the TableName is 'MIXED.'. When spatial references are returned (using the map, requesting a MapPoint or MapBox), the

Formatted: Table Row

Formatted: Table Row

			following information is present in the returned keylist: TableName : 'SPATIAL_REF' Item1, KeyField1 : Location Key, or Null if no location selected Item 1, KeyField2 : Spatial reference (top right if for a bounding box) Item 2, KeyField2 : Bottom left spatial reference for a bounding box.
ShowActiveForm	Procedure	IUnknown	Takes a GUID as a parameter. Creates an MDI child within Recorder 2000 and embeds a new instance of the ActiveX identified by the GUID onto the form. Can be used to create forms from within Addins without having to call menu options. The ActiveX control to be embedded must at least support IFormCaption or IRecorderAddin interfaces. The IUnknown return result is a pointer to the created ActiveX that is now displayed embedded on the MDI child. If this function is used alongside other interfaces such as IRecorderFormEvents, then 2 instances of the object are created by Recorder 2000. This means that methods such as SetForm will occur on one instance but not the other. The COM Object Factory used to instantiate the objects can be modified to allow singleton access to the addin, overcoming this problem. The ActiveX is also able to have a toolbar by implementing IFormWithToolbar.
Find	Function	WideString	Allows the Recorder 2000 find dialog to be used to locate Dictionary records. Takes the find dialog title, the type of search ('Taxon', 'Biotope' or 'Admin') and the initial search text as parameters. Returns the key string of the item found, or '' if none found.
GetNextKey	Procedure	WideString	Allows addins to generate new key values for a given table. Takes a table name as a parameter and returns the next key value in sequence. The LAST_KEY table is updated by Recorder

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

			2000.
MergeDatabase	procedure	None	Takes the path to an Access database as a parameter, and merges the contents of that database into the main database. Useful when adding new terms the database as part of an addin's installation procedure. Alternatively, a Zip file containing an mdb file may be specified.
CheckDatabase	procedure	None	Takes 2 parameters, the path to an Access database and a detail string. Opens the Import dialog and performs duplicate analysis/validation check on the database. Allows users to resolve issues and import the data. The detail string is displayed in the Details memo on the Import dialog. Alternatively, a Zip file containing an mdb file may be specified.
RecorderFunctions	Read only property	IRecorder Functions	Allows access to general purpose functions within Recorder 2000. See IRecorderFunctions for more information.
RecorderMainForm	Read only property	IRecorder MainForm	Allows access to functionality within the main form of Recorder 2000.
CanExportSystem Supplied	Read/write property	WordBool	Defaults to false. Allows addins to request that Recorder 2000 XML exports include system supplied data.
RecorderMap	Read only property	IRecorder Map	Returns an instance of IRecorderMap, allowing mapping functionality to be accessed.
Version	Read only property	WideString	Returns the version stamp of the Recorder 2000 application, for example '1.1.0.1'.
AccessSQLtoSQLServerSQL	Function	WideString	Takes a string that represents a valid SQL Statement in Access. Returns a string that is valid SQL Server syntax. If the SQL cannot be parsed then the string returns 'ERROR' on the first line. Note that not all Access 97 syntax is supported.
ConnectionString	Read Only property	WideString	Returns a connection string suitable for connecting to the nbndata database.
SetApplicationSecurity	Procedure	None	Takes an IConnection as a parameter. Initialises the correct application role for the connection according to the logged in user. This method must be called before using ADO

Formatted: Table Row

			Connections from addins. The connection should be opened first. The following code sample illustrates the use of this method and the ConnectionString property to initialise a Delphi TADOQuery: FQuery := TJNCCQuery.Create(nil); FConnection := TADOConnection.Create(nil); FConnection.LoginPrompt := False; FConnection.ConnectionString := FRec2k.ConnectionString; FConnection.Open; FRec2k.SetApplicationSecurity(FConnection.ConnectionObject); FQuery.Connection := FConnection;
ExportKeyList	Procedure	None	Takes 3 parameters, the KeyList to export, the export path, and the name of the export format type (valid formats are those listed in the Export Type combo box on the Data Export dialog, including Addin export formats). Exports the data to the supplied filename. If the file path or the export format type parameters are invalid, then the Data Export dialog is displayed.
ReportResults	Function	IReport Results	Returns a pointer to the IReportResults interface if the Wizard Results screen is visible, otherwise nil.
RequestCOMData	function	IUnknown	Functionality as RequestData, except that the data being requested is supplied by an addin implementing IKeyListSupplier. The method requires an GUID which indicates the addin screen data is being requested from, rather than a DataType string. Returns the addin class instance.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IRecorder6

The IRecorder6 interface descends from IRecorder2000 and is implemented within Recorder 6. It allows addins and external systems to display selected data specified by an SQL statement and display a specified tab in a screen.

Attribute	Type	Data Type	Description
DisplayDataFromSQL	Procedure	None	Takes two Widestring parameters. The first specifies the type of data to display (possible values are; 'Survey', 'Event', 'Sample', 'Taxon Occurrence', 'Biotope Occurrence', 'Location', 'Name', 'Location', 'Feature', 'Document', 'Taxon', 'Biotope' and 'Admin'). The second parameter specifies the SQL that selects the items for display in the screen. The SQL should return the key that uniquely specifies each data item. Result; Displays a Recorder data screen with the records specified by the SQL parameter listed in the tree view.

Attribute	Type	Data Type	Description
DisplayTab	Procedure	None	Takes six Widestring parameters. The first parameter specifies the type of data to display (possible values are; 'Survey', 'Event', 'Sample', 'Taxon Occurrence', 'Biotope Occurrence', 'Location', 'Feature', 'Name', 'Location', 'Document', 'Taxon', 'Biotope' and 'Admin'). The second parameter specifies the SQL that selects the items for display in the screen. The SQL should return the list of keys that uniquely specifies each data item. The third parameter specifies the SQL that selects the specific item to be highlighted in the list view and it's data displayed in the right hand of the screen. (the first item will be selected if a blank string or invalid key value is passed). The fourth parameter specifies the tab to display; this is defined using caption displayed on the tab in the Recorder interface. For nested tabs specify the sequence of tab captions, separated by slash characters ("\"), outermost tab first. If an empty string is passed then the default tab is selected. The fifth parameter is a Boolean value that indicates if the selected node is expanded in the tree view after the call to DisplayTab. This parameter is ignored if the selected node is not hierarchical (for example on the Documents screen). The sixth parameter is a Boolean value that indicates if the expansion is recursive (i.e. is the first level alone expanded, or does it expand all nodes beneath the selected node). The sixth parameter is ignored if the fifth parameter is false. NOTE: - A special case of the fifth and sixth parameters has been implemented. If the fifth parameter is false but the sixth is true then ALL main nodes and child nodes are expanded. Care should be exercised in using this

Attribute	Type	Data Type	Description
GetAvailableXmlReports	procedure	IXmlReportList	Takes a string parameter which is the keytype that XML Reports are being requested for. Possible values for this are: 'Name', 'Location', 'Taxon', 'Biotope', 'Admin', 'Survey', 'Event', 'Sample', 'Taxon Occurrence', 'Biotope Occurrence' or any type registered by an addin using the IReportKeytypeList interface. Returns a list of all available XML Reports that match the keytype requested.

IRecorderFunctions

This interface does not need to be implemented by add-in modules, just by Recorder 2000. Provides access to general purpose functions within Recorder 2000. For example, the following Delphi code illustrates conversion of HTML to RTF text:

```
var
  lRecorder2000 : IRecorder2000;
begin
  lRecorder2000 := CreateOleObject('Recorder2000.AutoApplicationSettings') as
                   IRecorder2000;
  RtfText := lRecorder2000.RecorderFunctions.HTMLtoRTF( Html );
end;
```

The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
HTMLtoRTF	Read only property	Widestring	Converts the supplied string (in HTML format) to the equivalent string in RTF format. Supports the following HTML items only: bold, italic, underline, emphasis, strong emphasis, named and numeric character entities (ISO 8859-1), comments
SpatialRefSystem	Read only property	Widestring	Returns the 4 character code identifying the currently selected spatial reference system.
DecodeSpatialRef	Function	ILatLong	Returns latitude and longitude of the spatial reference string passed in. Recorder 2000 attempts to identify the spatial reference system in use, where this is not possible an error is

Formatted: Table Row, Justified

Formatted: Table Row, Justified

Formatted: Table Row, Justified

Formatted: Table Row, Justified

			returned in ILatLong.ErrorMsg.
GetSpatial RefFromRecord	Function	Widestring	Takes two parameters, an IKeyList which locates a single record in the database, and a field specifier. The field specifier is only required where the database record identified contains 2 or more spatial references, for example this contains 'NE' to identify the North East corner of the bounding box in the SURVEY table. The return value is a spatial reference string, formatted for display according to the currently selected spatial reference system.
DecodeVagueDate	Function	IVagueDat e	Takes a widestring parameter, which is a vague date display string. Decodes it into a start date, end date and date type.
EncodeVagueDate	Function	WideString	Takes an IVagueDate parameter and converts it into a display string, for example '01 Jun 2000', '31 Aug 2000', 'P' is converted to 'Summer 2000'.
GetDateFrom Record	Function	WideStrng	Takes an IKeyList parameter which identifies the record for which a date string should be returned. A second parameter identifies the prefix of the field required when 2 or more date fields are present in the record. This function only returns vague dates from records. For example, when a survey event record is queried the date returned is the event date. When a survey record is queried it is necessary to specify 'from' or 'to' in the specifier to identify which date is required. Dates returned are strings in a suitable format for display, e.g. 'Spring 1985'.
EncodeSpatialRef	Function	WideString	Takes an ILatLong parameter. Encodes it into suitable display string depending on the currently selected reference system.
IdentifySpatialRef System	Function	WideString	Takes a widestring parameter that represents a spatial reference. The return result is the identified reference system (e.g. 'OSGB'), or 'Grid System Unknown' if the spatial reference is not recognised.
GetFilterItems	Function	IKeyList	Returns a key list identifying records that match the filter identified by the

Formatted: Table Row, Justified

			EXPORT_FILTER_KEY supplied as a parameter. Valid keys are those listed in the EXPORT_FILTER table.

IRecorderMainForm

This interface does not need to be implemented by add-in modules, just by Recorder 2000. Allows addins to access the main form of Recorder 2000. For example. the following Delphi code sets the status bar texr:

```
var
  lRecorder2000 : IRecorder2000;
begin
  lRecorder2000 := CreateOleObject('Recorder2000.AutoApplicationSettings') as
                   IRecorder2000;
  lRecorder2000.RecorderMainForm.StatusText := 'Analysing data…';
end;
```

Attribute	Type	Data Type	Description
StatusText	Read/write property	WideString	Allows the status bar text to be read or written to.
StartProgressBar	Function	None	Takes control of the Recorder 2000 progress bar and sets the position to zero. This prevents Recorder 2000 from setting the progress bar so it must be followed by a cell to StopProgressBar.
Progress	Read/write property	Integer	Sets the progress bar position to a percentage position. Values below zero and above 100 are ignored, and writing to this property has no effect unless preceded by a call to StartProgressBar.
StopProgressBar	Function	None	Releases control of the Recorder 2000 progress bar and sets the position to zero.

Formatted: Table Row, Justified

IRecorderMap

This interface does not need to be implemented by add-in modules, just by Recorder 2000. Allows access to Recorder 2000's map functionality.

Attribute	Type	Data Type	Description
Scale	Property	WideString	Allows addins to determine the current scale of the map. This is achieved by returning a a string containing comma-separated values of the coordinates (in lat-long) of three points on the map. The coordinates of these points in

Formatted: Table Row, Justified

			pixels are (0,0), (0,3) and (3,0), relative to the top left corner of the map window.
PanMap	Function	None	Takes a WideString parameter. Displays the map form, and pans the map so that the supplied spatial reference is centred in the window. The spatial reference string is interpreted within the application to determine the system, if the system cannot be determined then an error is displayed.
BaseMapRefSystem	Read Only property	WideString	Returns a 4 digit string identifying the base map's coordinate system, e.g. 'OSGB'.
RefreshSheets	Function	Boolean	If the Map Window is visible, then refreshes the map and its visible sheets, and the return value is True. If the Map Window is not currently visible, then False is returned.

Formatted: Table Row, Justified

Formatted: Table Row, Justified

Formatted: Table Row, Justified

Addins are able to configure the sheets visible on the map by editing the MAP_SHEET table then calling RefreshSheets. The following table describes the fields in the MAP_SHEET table which are edited to configure sheets on the map:

COLUMN NAME	DESCRIPTION
MAP_SHEET_KEY	Unique identifier for the MAP_SHEET table. Required. When adding a new sheet, populate this with the next NBN Key value for the MAP_SHEET table.
USER_ID	Identifies the user to which the map sheet applies to. Foreign key to the NAME_KEY field in the NAME table. This is specified ONLY for Base Maps and Background Layers (Sheet Type 0, 1 or 2). Polygon layers are applicable to all users and this field is left null.
SHEET_NAME	Name of the map sheet.
FILE_NAME	Filename for the map sheet. Populate this with the source file name for the sheet, and full path. If adding a new sheet as a background layer, then this is the raster or vector file to import. If adding a polygon layer, then construct a unique 'gsf' filename in the Object Sheet directory, but do not actually create the file as the Map will create it when refreshed. It is recommended that the file name is constructed using the NBN key for the reqcord, thus guaranteeing uniqueness.
SHEET_TYPE	Indicates the type of sheet : 0 = Base Maps, 1 = Vector background layer, 2 = Raster background layer, and 3 fo Polygon layers.

	Required.
SW_SPATIAL_REF	Spatial reference of the southwest corner of the bounding box for the map sheet. Not Required. Applies to background map layers (type 1 or 2) only.
NE_SPATIAL_REF	Spatial reference of the northeast corner of the bounding box for the map sheet. Not Required. Applies to background map layers (type 1 or 2) only.
SPATIAL_REF_SYSTEM	Original system used to record the spatial reference. Not Required. Applies to background map layers (type 1 or 2) only.
SW_LAT	Latitude of the southwest corner of the bounding box for the map sheet. Stored as double precision floating point. Not Required. Applies to background map layers (type 1 or 2) only.
SW_LONG	Longitude of the southwest corner of the bounding box for the map sheet. Stored as double precision floating point. Not Required. Applies to background map layers (type 1 or 2) only.
NE_LAT	Latitude of the northeast corner of the bounding box for the map sheet. Stored as double precision floating point. Not Required. Applies to background map layers (type 1 or 2) only.
NE_LONG	Longitude of the northeast corner of the bounding box for the map sheet. Stored as double precision floating point. Not Required. Applies to background map layers (type 1 or 2) only.
NE_SPATIAL_REF_QUALIFIER	Qualifier for the NE spatial reference. Not Required. Applies to background map layers (type 1 or 2) only.
SW_SPATIAL_REF_QUALIFIER	Qualifier for the SW spatial reference. Not Required. Applies to background map layers (type 1 or 2) only.
CUT_IN_SCALE	The scale at which the map sheet becomes visible. Not required. For example '1:100000'.
CUT_OUT_SCALE	The scale at which the map sheet becomes invisible. Not required. For example '1:1000'.
SHEET_DISPLAYED	Indicates whether or not the sheet is visible in the map browser. Not required.
NEW_DATA	Indicates whether or not the sheet is new and therefore needs importing/attaching to the dataset for the map. Not required. Set this flag when adding a new sheet record to the map. When the user opens the map, an automatic scan is performed to pick up new layers, so this flag is not relevant. However, while the

	map is iin use, addins can set this flag and call RefreshSheets to trigger detection of the layer.
MODIFIED_DATA	Indicates whether or not the sheet associated with the record needs to be modified with regards to it's cut in/ cut out point and extents. Not required. Set this flag when the modifying the configuration of an already existing sheet.
REMOVE_SHEET	Indicates whether or not the sheet is to be removed from the dataset. Not required. Set this flag when a sheet is to be deleted (the Map will delete this record automatically when refreshed).
DATASET_SHEET_NAME	Name that the sheet is referred to as in code. Do not alter this value.
DATASET_SHEET_FILENAME	Filename of the MapServer sheet created from the file specified in FILE_NAME. Do not alter this value.
ENTERED_BY	User who originally entered the record. Refers to an item in the NAME table but does not use a physical Access relationship. Required.
ENTRY_DATE	Date of entry into the system for the record. Defaults to Now(). Required.
CHANGED_BY	User most recently changed the record. Refers to an item in the NAME table but does not use a physical Access relationship. Not Required.
CHANGED_DATE	Date of most recent edit for the record. Not Required.
UNSELECTED_COLOUR	RGB value of the unselected colour for this sheet. Only applicable is this is a polygon layer (SHEET_TYPE=3). The calculation is (red value) * 256 ^ 2 + (green value) * 256 + (blue value)
SELECTED_COLOUR	RGB value of the selected colour for this sheet. Only applicable is this is a polygon layer (SHEET_TYPE=3).
PATTERN_INDEX	Index of the pattern used for this layer. Corresponse to the ordinal value of the equivalent TBrushStyle type in Delphi. Only applicable is this is a polygon layer (SHEET_TYPE=3). The following list of patterns are supported: 0 = Solid 1 = Clear 2 = Horizontal 3 = Vertical

	4 = Backward Diagonal 5 = Forward Diagonal 6 = Cross 7 = DiagCross

ICurrentSettings

This interface does not need to be implemented by add-in modules, just by Recorder 2000. Allows the current settings of Recorder 2000 to be passed across COM to Add-In modules. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
TaxonListKey	Read Only Property	WideString	Contains the 16 character unique ID for the currently selected taxon list in the taxon dictionary.
BiotopeListKey	Read Only Property	WideString	Contains the 16 character unique ID for the currently selected biotope classification in the biotope dictionary.
AdminAreaListKey	Read Only Property	WideString	Contains the 16 character unique ID for the currently selected admin area type in the admin dictionary.
ResultList	Read Only Property	IKeyList	Contains a mixed list of 16 character unique IDs for taxon and biotope occurrence records.
UserIDKey	Read Only Property	WideString	Identification key for the current user. Should be used to populate the Entered_by and Changed_by fields when populate data in a COM addin.

Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Space Before: 6 pt, After: 6 pt, Keep lines together
Formatted: Table Row
Formatted: Space Before: 6 pt, After: 6 pt, Keep lines together
Formatted: Table Row
Formatted: Space Before: 6 pt, After: 6 pt, Keep lines together

ICurrentSettings6

Version of the ICurrentSettings interface implemented in Recorder 6. Inherits all the existing attributes of ICurrentSettings and adds the following:

Attribute	Type	Data Type	Description
DragDestinationColour	Read only property	Integer	Provides access to the color used for drag destination frames in Recorder. This information is current, as opposed to registry

Formatted: Table Row
Formatted: Table Row

			information which is only updated when the applicatio is closed.
DragSourceColour	Read only property	Integer	Provides access to the color used for drag source frames in Recorder. This information is current, as opposed to registry information which is only updated when the applicatio is closed.
MandatoryColour	Read only property	Integer	Provides access to the color used for mandatory fields in Recorder. This information is current, as opposed to registry information which is only updated when the applicatio is closed.
DisableDragDropFrames	Read only property	Wordbool	Provides access to setting dictating if drag frames are visible in Recorder. This information is current, as opposed to registry information which is only updated when the applicatio is closed.
AvailableMapCount	Read only property	Integer	Retrieves the number of base maps that have been setup and are available.
AvailableMap	Read only property	IAvailable Map	Retrieves information for a particular base map that is available.
DefaultMap	Read only property	IAvailable Map	Retrieves information about the users default base map.
SessionID	Read only property	String	Retrieves the current session ID. This uniquely identifies the login session in the Session table.

Formatted: Table Row

IRucksack

This interface does not need to be implemented by add-in modules, just by Recorder 2000. Allows rucksack objects to be passed across COM from Recorder 2000 to Add-In modules. An interface pointer to the Rucksack object is passed to all addins using the IRecorder2000 interface. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
TaxonList	Read Only Property	IKeyList	Contains a list of 16 character unique IDs for the TAXON_LIST_ITEM table.
BiotopeList	Read Only	IKeyList	Contains a list of 16 character unique IDs for the

Formatted: Table Row

	Property		BIOTOPE_LIST_ITEM table.
LocationList	Read Only Property	IKeyList	Contains a list of 16 character unique IDs for the LOCATION table.
NameList	Read Only Property	IKeyList	Contains a list of 16 character unique IDs for the NAME table.
DocumentList	Read Only Property	IKeyList	Contains a list of 16 character unique IDs for the REFERENCE table.
RucksackFile	ReadOnly Property	Widestring	File name and path of the current rucksack. Blank if the rucksack is not saved.

Formatted: Table Row
Formatted: Table Row
Formatted: Space Before: 6 pt, After: 6 pt, Keep lines together
Formatted: Table Row
Formatted: Table Row

IRecorderForm

Provides access to a Recorder form to an external addin. Addins must implement the IRecorderFormEvents to receive an interface pointer to IRecorderForm. Once the interface pointer is available, addins can access controls and embed controls onto standard forms.

Attribute	Type	Data Type	Description
Control	Read only property	IUnknown	Takes the name of the required control (WideString) as a parameter. Returns a COM object to the addin which implements IRecorderControl, plus any other control interface depending on the control type. For example, a TEdit control may be requested by name and implements IRecorderControl and IEditControl.
EmbedActiveX	Procedure	None	Allows an addin to embed ActiveX components onto existing Recorder forms. The following parameters are required: iControl (iUnknown – pointer to the ActiveX instance) iClsID (GUID – identifier for the ActiveX class iParent (IRecorderControl – control on which the control should be embedded. If nil, then the control is embedded directly onto the form.) iLeft, iTop, iWidth and iHeight are all integer parameters which specify the control's position.

Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

IRecorderControl

Returned by the Control property of IRecorderForm. Provides basic information for any control on a Recorder form, and allows external addins to modify forms (for example existing controls can be hidden or moved, and new ActiveX controls embedded).

Attribute	Type	Data Type	Description
Name	Read only property	WideString	The name of the control.
Left	Read / write property	Integer	The left position (in pixels) of the control.
Top	Read / write property	Integer	The top position (in pixels) of the control.
Width	Read / write property	Integer	The width (in pixels) of the control.
Height	Read / write property	Integer	The height (in pixels) of the control.
Visible	Read / write property	WordBool	Flag indicating that the control is visible.
Enabled	Read / write property	WordBool	Flag indicating that the control is enabled.
RegisterEvent Handler	Procedure	None	Takes the name of a Delphi event, and an object that implements IEventHandler, as parameters. The event must be a Delphi TNotify event which is available for the control, for example 'OnClick'. Causes the event to trigger the OnEvent method of the supplied IEventHandler object, allowing existing controls' events to be trapped by COM addins.

Formatted: Table Row

IEditControl

Provides additional properties for controls which are descendants of Delphi's TCustomEdit, and also TDBGlyphLookupComboBox. TDBGlyphLookupComboBox does not operate as a combo box over COM because it does not support an Items array,

or an ItemIndex. It supports a read-only text property (setting a value to text has no effect).

Attribute	Type	Data Type	Description
Text	Read / write property	WideString	The text of the edit control.

Formatted: Table Row

IListControl

Provides additional properties for controls which are descendants of Delphi's TCustomListBox.

Attribute	Type	Data Type	Description
ItemCount	Read only property	Integer	Number of items in the list box
Items	Read / write property	WideString	Provides access to the text of each item in a list box. The index of the item (integer) is supplied as a parameter.
SelectedIndex	Read / write property	Integer	Item Index (starting at zero) of the currently selected item.
Selected	Read only property	WordBool	Takes an index of an item as a parameter. Returns true if the item is selected – can be used to read all selected items from a multi select list box.

Formatted: Table Row

IComboControl

Provides additional properties for controls which are descendants of Delphi's TCustomComboBox.

Attribute	Type	Data Type	Description
ItemCount	Read only property	Integer	Number of items in the combo box
Items	Read / write property	WideString	Provides access to the text of each item in a combo box. The index of the item (integer) is supplied as a parameter.
SelectedIndex	Read / write	Integer	Item Index (starting at zero) of the currently selected item.

Formatted: Table Row

	property		

IGridControl

Provides additional properties for controls which are descendants of Delphi's TStringGrid.

Attribute	Type	Data Type	Description
ColumnCount	Read only property	Integer	Number of columns in the grid
RowCount	Read only property	Integer	Number of rows in the grid
CurrentColumn	Read only property	Integer	Index of the current cell's column (starting at zero)
CurrentRow	Read only property	Integer	Index of the current cell's row (starting at zero)
Cells	Read / write property	WideString	Returns the string stored in the cell identified by the iX and iY integer parameters.

Formatted: Table Row

ILabelControl

Provides additional properties for controls which are descendants of Delphi's TCustomLabel.

Attribute	Type	Data Type	Description
Caption	Read / write property	WideString	Provides access to the caption of a label.

Formatted: Table Row

ITabPageControl

Provides additional properties for controls which are descendants of Delphi's TTabSheet control.

Attribute	Type	Data Type	Description
Caption	Read / write property	WideString	Provides access to the caption of a tab sheet.

Formatted: Table Row

IReportResults

Provides access to attributes of the Wizard Results screen.

Attribute	Type	Data Type	Description
ReportConnection	Read only property	Connection	Provides access to the ADO Connection on which the report's underlying temporary table exists. Temporary tables are only accessible within the connection that created them.
ReportSQL	Read only property	String	Provides access to the SQL that was used to populate the results grid, including order by information.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

The following example code illustrates a simple example of the usage of the IReportResults interface to generate a CSV file.

```
procedure TReportToCsv.OutputReport(const AFilePath : string);
var
  lConnection : TADOConnection;
  lqryReport : TAdoQuery;
  lOutput : TStringList;
  lRec2K : IRecorder2000;
  lOldConnection : _Connection;
begin
  lRec2k := CreateOleObject('Recorder2000.AutoApplicationSettings') as
   IRecorder2000;
  if lRec2k.ReportResults = nil then
    ShowMessage('No report available')
  else begin
    lConnection := TADOConnection.Create(nil);
    lqryReport := TADOQuery.Create(nil);
    try
      lOldConnection := lConnection.ConnectionObject;
      lConnection.ConnectionObject := lRec2k.ReportResults.ReportConnection as
                                      AdoInt._Connection;
      lqryReport.Connection := lConnection;
      lqryReport.SQL.Text := lRec2k.ReportResults.ReportSQL;
      lqryReport.Open;
      lOutput := TStringList.Create;
      while not lqryReport.EOF do begin
        lOutput.Add(GetRecordAsCsv(lqryReport));
        lqryReport.Next;
      end;
      lOutput.SaveToFile(AFilePath);
      lqryReport.Close;
    finally
      lqryReport.Free;
      // remove the reference to the recorder connection, otherwise freeing the
      // lConnection closes the one in Recorder!
      lConnection.ConnectionObject := lOldConnection;
      lConnection.Free;
    end;
  end;
end;
```

```
function TReportToCsv.GetRecordAsCSV(AQuery : TADOQuery) : string;
var
  i : integer;
begin
  Result := '';
  for i := 0 to AQuery.Fields.Count-1 do
    if Pos(',', AQuery.Fields[i].Value) > 0 then
      AddToCommaSeparatedList(Result, '"'+AQuery.Fields[i].AsString + '"')
    else
      AddToCommaSeparatedList(Result, AQuery.Fields[i].AsString);
end;
```

IAvailableMap

Provides access to attributes of a base map that has been initialised by the user.

Attribute	Type	Data Type	Description
Title	Read only property	String	Display title of the base map.
Key	Read only property	String	Base_Map_Key for the Base_Map record.
IsDefault	Read only property	Wordbool	Flag indicating if the map is the user's default map.
SpatialSystem	Read only property	String	4 character spatial reference system code for the system used by this base map.
Display	Procedure	None	Allows the addin to call for the map to be displayed in Recorder.
DisplayDistribitionPoints	Procedure	None	Allows an addin to supply a list of distribution points to Recorder for display on a specific map. Recorder first displays the map if it is not already visible, then loads the dataset. Takes 2 parameters, the first is the dataset title, the second is the list of map points (using the IMapDropFormat interface to transfer the map points to Recorder).

Formatted: Table Row

XML Report List Interface

Interface allowing Recorder to describe a list of available Xml Reports.

IXMLREPORTLIST INTERFACE			
ATTRIBUTE	**TYPE**	**DATA TYPE**	**DESCRIPTION**
ReportCount	Read only property	Integer	Number of reports in the list.
Report	Read only property (indexed)	IXmlReport	Provides access to each individual report in the list.

Formatted: introduction

XML Report Interface

Interface allowing Recorder to describe a single Xml Report.

IXMLREPORT INTERFACE			
ATTRIBUTE	**TYPE**	**DATA TYPE**	**DESCRIPTION**
ReportTitle	Read only property	string	Title of the report.
ReportPath	Read only property	string	Hierarchical folder path of the report, with '\' separating each level. For example "Species Reports\Statistics" equates to a top level folder called Species Reports with a subfolder called Statistics.
Execute	procedure	Boolean	Allows an addin to trigger an Xml Report. Takes a single IKeyList parameter which defines the data items to be reported on. The IKeyList should contain keys of the correct type for the keytype of the report. The keylist should contain only one item unless the keytype defined by an addin has been flagged as supporting multiple values (Multivalues=true). Returns true if the report was run, or false if the report operation was cancelled.

Formatted: introduction

Interfaces implemented by Addins

IRecorderAddin

This interface is implemented by all addin classes which are compatible with Recorder2000. It provides generic information such as the name and description, and provides initialisation methods.

Attribute	Type	Data Type	Description
Name	Readonly property	WideString	Name of the addin class. Used as a descriptive name for the addin within Recorder 2000.
Description	Readonly property	WideString	Description of the addin class.
ImageFileName	Readonly property	WideString	Filename of a 16*16 bitmap file which is associated with the addin. This file is placed in a folder '\Addins\Images\' under the application directory during installation, and must be located within the same folder as the addin library (ocx or dll) file when installation of the library starts. The file's transparent colour is automatically read from the bottom-left most pixel of the bitmap.
Install	Procedure	None	Allows the class to perform any installation required, e.g. adding new termlist. Note that there is no equivalent uninstall procedure as this may remove items required by other users of a network installation who are still using the addin. The iInstallFilePath parameter provides the complete file path of the addin being installed. Therefore it is possible to locate other files associated with the addin during the installation process.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: No underline

IHelp

Classes which implement IHelp can provide a Help contents page for any addin. The class is responsible for initiating its own Help system, which therefore can potentially include specially written tutorials. Addin forms may pick up their own context sensitive help if required.

Recorder automatically creates a menu item on the Help menu for any addin which implements IHelp. The exception is new tab page addins (IAdditionalPage

implementations) in which case the help method is invoked when the active control is on the tab and the user presses the F1 key.

Attribute	Type	Data Type	Description
DoHelpContents	Procedure	None	Initiates the display of the Help contents page. The addin is responsible for ownership of the help form, which can therefore be any suitable help system.

IValidation

This interface is used for validating a set of newly imported data. It performs validation on an entire database basis and returns items which fail and cannot be imported.

Additional form based validation can be provided by implementing IRecorderFormEvents.

Attribute	Type	Data Type	Description
Validate	Function	Integer	Takes a database path and name as a parameter. The object supporting this interface then checks all data in the given database for validation failures. If failures are located, the return result is the number of failures, otherwise Zero.
KeyList	Function	IKeyList	Allows the validation class to return the list of failed records for the last call to V~~v~~alidate (or ValidateAll or ValidateKeyList if IValidation6 is implemented). If records fall into more than one table, then the KeyField2 of the key item can be used to store the Table name of the record, whilst the lists tablename is set to 'MIXED'.
ErrorString	Function	WideString	Takes an iIndex integer as a parameter. Returns the error string associated with a validation failure reported using KeyList. The iIndex parameter identifies the item in the corresponding position within the key list.

IValidation6

Inherited from IValidation and allows validation libraries to implement additional functionality available in Recorder 6.12 or later. The additional functionality is oriented towards allowing the validation library to validate parts or all of the main database rather than a temporary import database.

Attribute	Type	Data Type	Description
ValidateAll	Function	Integer	Recorder calls this method to request that the validation addin validates the entire content of the SQL Server NBNData database.
ValidateKeyList	Function	Integer	Recorder calls this method to request that the validation addin validates the content of the SQL Server NBNData database as identified by the keylist supplied. Note that when calling this method, the validation library does not need to expand nodes to find the children – this is the responsibility of the caller when required. For example if a request to validate a survey is issues, then only the survey should be validated, not the contained events, samples and occurrences.

Formatted: English (U.K.)
Formatted: Table Row
Formatted Table
Formatted: English (U.K.)
Formatted: Table Row
Formatted: Table Row
Formatted: Normal

IFormCaption

IFormCaption does not specify a Recorder addin in its own right, but is optionally implemented alongside any other interface which specifies a form. Form addins are actually implemented by embedding ActiveX controls directly onto a blank Recorder form, so the addin is not in direct control of the form's properties such as its caption. This interface allows the caption of the form to be set, otherwise the default or Addin name are used.

Attribute	Type	Data Type	Description
FormCaption	Read only property	WideString	Specifies the caption of the container form onto which the ActiveX addin is placed.

Formatted: Table Row
Formatted: Table Row

IReplacementForm

Classes which implement IReplacementForm must be ActiveX controls. This interface allows an ActiveX to replace an existing data entry screen as an MDI child form, by embedding the activeX onto a blank form within Recorder2000. The form's caption is set to the name of the addin (from IRecorderAddin), or the name specified in the IFormCaption interface if specified.

Classes which implement IReplacement form may also implement IKeyListSupplier, for a greater level of integration with existing Recorder functionality.

The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description

Formatted: Table Row

FormToReplace	Read only Property	WideString	The name of the data entry screen the Active Form is to replace. See below for list of available forms to replace.

Formatted: Table Row

Available Recorder 2000 forms to replace
TfrmPlaceCard
TfrmSpeciesCard
TfrmObservations
TfrmSurveyDetails
TfrmEventDetails
TfrmSampleDetails
TfrmTaxonOccurrences
TfrmBiotopeOccurrences
TfrmLocations
TfrmLocationDetails
TfrmFeatureDetails
TfrmIndOrg
TfrmReferences
TfrmBiotopeDictionary (deprecated, use TfrmBiotopeDictBrowser)
TfrmBiotopeDictBrowser
TfrmTaxonDictionary (deprecated, use TfrmTaxonDictBrowser)
TfrmTaxonDictBrowser
TfrmAdminAreaDictionary (deprecated, use TfrmAdminAreaDictBrowser)
TfrmAdminDictBrowser
TfrmTermLists
TfrmRucksack
TfrmResultWindow
TfrmMap
TfrmTaxonDictDetails (deprecated, use TfrmTaxonDictEditor)
TfrmTaxonDictEditor
TfrmBiotopeDictDetails (deprecated, use TfrmBiotopeDictEditor)
TfrmBiotopeDictEditor

IKeyListSupplier

Addins which implement IReplacementForm or INewAction may also implement IKeyListSupplier, but not if they implement IDialog or IExecuteAction. This allows the addin to provide Key lists to other aspects of the software. Key lists are simply lists of records identified by key values, and facilities such as Data Export and Return Data require them. All replacement forms that support IKeyListSupplier can be used to Return Data, and can optionally be used as sources for Data Export. Classes implementing IKeyListSupplier must be able to supply additional classes which implement IKeyList and IKeyItem.

Keylists can contain data from multiple tables. This is implemented by returning 'MIXED' in the keylist's table name, and by returning actual table names in the KeyField2 of each item on the keylist.

Attribute	Type	Data Type	Description
KeyList	Read only property	IKeyList	Returns a key list indicating the currently selected data in the form. For example, a replacement for frmLocations will normally return a keylist containing a single item – the currently selected location.
Exportable	Read only property	WordBool	Indicates if the KeyList provided may be used for data export. If so, then the Data Export menu option is displayed as appropriate.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IRequestor

IRequestor allows an addin to be notified of Return Data operations which were requested by the COM Addin. IRequestor can by implemented directly within the addin ActiveX, or can by implemented within a secondary object specifically for the purpose. The IRequestor is supplied to Recorder 2000 via the IRecorder2000.RequestData method.

Attribute	Type	Data Type	Description
Update	Procedure	None	Takes a single IKeyList parameter. This provides the addin with the data items returned from the Return Data function.

Formatted: Table Row

Formatted: Table Row

IEventHandler

IEventHandler is implemented by COM addins that want to be notified of an event occurring on a control in Recorder 2000.

Attribute	Type	Data Type	Description
OnEvent	function	boolean	Takes the name of the control and the name of the event which occurred as parameters. Therefore it is possible to implement 1 IEventHandler object to handle many events for many controls. Allows the addin to respond to the event. Return true to allow the normal event handling in Recorder 2000 to occur, and false, to prevent it (this should only be used with care!). See IRecorderControl for more information.

Formatted: Table Row

Formatted: Table Row

IAdditionalPage

This interface allows ActiveX controls / Active Forms to add themselves to existing edit screens as an extra detail page (tab sheet) at the end of the page control. Additional pages can only access database records other that those already visible on the main form due to locking restrictions of Access.

ActiveX controls which implement IAdditionalPage can receive notifications of changes to form state by implementing IRecorderFormEvents.

If the CurrentTable and CurrentKey properties are set to '', then the additional page should reset all its controls to blank.

The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
Form	Read only Property	WideString	The class name of the edit screen the ActiveX control / Active Form attachs to. (e.g. TfrmLocations)
PageCaption	Read only Property	WideString	The caption for the new tab sheet that houses the ActiveX control / Active Form.
CurrentTable	Read/Write property	WideString	Allows the additional page to be notified of the table which it is supposed to link to.
CurrentKey	Read/Write property	WideString	Allows the additional page to be notified of the record which it is supposed to link to. The additional page cannot access data within that table, but can use the key value to identify records in any addin tables which need to be displayed.

Formatted: Table Row
Formatted: Table Row
Formatted: English (U.K.)
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

IAdditionalPage6

This interface is implemented in Recorder version 6.9.3. It allows the addin to specify the sequence of the additional tabs created using IAdditionalPage.

Attribute	Type	Data Type	Description
TabIndex	Read only property	Integer	Position of the page relative to the other tabs (zero indexed).

Formatted: Table Row
Formatted: Table Row

INewAction

This interface allows COM object to declare actions so that they may be accessed via the menu bar and tool bar of frmMain.

New actions provide additional dimmed and disabled images as well as the image provided by IRecorderAddin.

By default, new actions create an MDI child form onto which the action class ActiveX is embedded. However, if the same class implements IDialog then the action is displayed as a modal dialog, or if it implements IExecuteAction then the action simply triggers an execute method. MDI child actions (which don't implement IDialog or IExecuteAction) can optionally implement IKeyListSupplier. This allows the standard Return Data and Export functionality of Recorder 2000 to be utilised.

The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
ActionCaption	Read only Property	WideString	The caption that will be displayed when the action is assigned to the menubar of frmMain.
Hint	Read only Property	WideString	The hint that will be displayed in the status bar of frmMain when menu items or toolbar buttons linked to this action are "hovered over".
DimmedImage FileName	Read/write Property	WideString	The filename where the dimmed image displayed on menu items or toolbar buttons linked to this action can be found. When users install COM objects supporting this interface property they will be required to provide a *.BMP file which will be placed in a sub-directory of the application's installation directory. Can be null.
DisabledImage FileName	Read/write Property	WideString	The filename where the disabled image displayed on menu items or toolbar buttons linked to this action can be found. When users install COM objects supporting this interface property they will be required to provide a *.BMP file which will be placed in a sub-directory of the application's installation directory. Can be null.
ParentMenu	Read only Property	WideString	Contains the name of the Menu Item to which the new action is added. If not already present, then new menu is created.
CanAddToToolbar	Read only property	WordBool	Can only be set to true if DimmedImageFileName doesn't equal the Null string, '', indicates whether or not the action can be added to the main toolbar via the Main Toolbar page of the Options screen.

Formatted: English (U.K.)
Formatted: Table Row
Formatted: English (U.K.)
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

IDialog

This interface is implemented in classes which implement either INewAction (in which case the action displays a modal dialog) or IAdditionalFilterDialog. It allows the class (which must be an ActiveX) to declare its dimensions, and to handle clicking of the Ok and Cancel buttons on the dialog.

Classes which implement IDialog are embedded onto a dialog which already has Save and Cancel buttons. The ActiveX control is embedded into the top part of the dialog above the buttons.

Attribute	Type	Data Type	Description
Width	Read only Property	Integer	Returns the width required to display the ActiveX dialog control.
Height	Read only property	Integer	Returns the height required to display the ActiveX dialog control. Does not include the height of the Ok and Cancel button panel.
DoOk	function	WordBool	Trigerred when the dialog OK button is clicked – allows the Addin to respond to the Ok button. The Addin may prevent the dialog from closing (if there is a validation failure for example) by returning false.
DoCancel	procedure	None	Trigerred when the dialog Cancel button is clicked – allows the Addin to respond to the Cancel button.

Formatted: Table Row

IExecuteAction

New actions which implement this are not displayed as activeX controls on either a MDI form or a modal dialog. The action is triggered as a non-visual procedure when the menu option is selected, and it can subsequently perform any required operations.

Attribute	Type	Data Type	Description
Execute	procedure	None	Initiates execution of the action procedure – called when the action's menu option is clicked.

Formatted: Table Row

IFormWithToolbar

Implementing this interface allows any COM form displayed as an MDI child to have its own toolbar.

It is recommended that the Toolbar ActiveX is returned as a separate ActiveX class. The ActiveX exposes a read/write property, LinkedForm, which enables the COM addin

to pass a reference to itself to the toolbar. This enables the toolbar ActiveX to call methods on the COM addin's form.

Attribute	Type	Data Type	Description
Toolbar	read only property	IUnknown	Returns an instance of an ActiveX which is to be embedded onto Recorder 2000's control bar as the COM form's active toolbar.

Formatted: Table Row

Formatted: Table Row

IFilter

Recorder 2000 allows the specification of new file format filters by implementing the IFilter interface. In addition, the class implements IImportFilter and/or IExportfilter to determine that the filter is available for import or export. In addition, classes which implement IExportFilter may also implement IExportByColumn, indicating that the export filter requires a particular measurement item to export data for (for example, the user can specify to export wing length data to a GIS).

Attribute	Type	Data Type	Description
DefaultFileExtension	Read only Property	WideString	The file extension displayed in the open dialog called from dlgDataImport.

Formatted: Table Row

Formatted: Table Row

IImportFilter

This interface allows a COM object to import data from any file data source that contains data adhering to the NBN data model. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
ImportFile	Function	Boolean	Takes a Widestring parameter which indicates the path of the file to import. Attempts to import the data into the database. Returns True if successful and False otherwise.
LastImportError	Read only Property	Widestring	If a call to ImportFile results in a failure, Recorder2000 reads this property to determine the error string to display.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IExportFilter

This interface allows a COM class to define a new export file format for NBN data. Classes can implement both IImportFilter and IExportFilter. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
ExportFile	Function	Boolean	Takes a IKeyList parameter which indicates the data to export, and a second WideString parameter which indicates the file location to export to. Returns true if the export file is successfully created and False otherwise.
LastExportError	Read only Property	Widestring	Only set if Export = False, returns the error of the last record to fail the export process.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IExportFilter6

This interface is implemented in Recorder version 6.12.2.149. It allows the addin to process records that have failed revalidation before export.

Attribute	Type	Data Type	Description
ExportFileWithFilteredInvalids	Function	Boolean	Takes a IKeyList parameter which indicates the data to export, a IKeyList which indicates the items that have failed revalidation, and a WideString parameter which indicates the file location to export to. Returns true if the export file is successfully created and False otherwise.

Formatted: Table Row

Formatted: Table Row

IExportByColumn

Allows a data export to request a measurement unit and type to export. If implemented, then Recorder 2000 displays a dialog allowing the user to select one of the existing measurement types, and the results of this selection are passed to the export instance before ExportFile is called.

Attribute	Type	Data Type	Description
MeasurementUnitKey	Read / Write Property	WideString	The key value of the measurement unit (which defines the measurement type) to export. This is set before a call to ExportFile is made.
NumberOfRanges	Read / Write Property	Integer	Allows the user to set the number of ranges (2-9) data recorder against the selected measurement unit will be sampled against.
MinValue	Read / Write Property	Double	Allows the user to set the minimum value for the data sample.
MaxValue	Read / Write	Double	Allows the user to set the maximum value for

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

	Property		the data sample.

ISpatialReference

This interface allows a COM object to add a new grid reference system to Recorder 2000 to allow recordings to be made against. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
Name	Read only Property	WideString	The name that the grid reference system is displayed as in the "Spatial References" page of dlgOptions.
SpatialRefSystem	Read/write Property	Integer	The global identifier for the grid reference system. This is picked up from the setting on the "File Locations" page of dlgOptions. By default, "OS GB National grid" is "OSGB", "OS Irish Grid" is "OSNI", "Latitude and Longitude" is "LTLN" and "UTM" is "UTM". Each installed COM object supporting this interface will have this property set as a unique 4 character string. Third parties wishing to implement a new spatial reference system must obtain a unique identifier from JNCC.
ConvertToLat	Function	WideString	Takes a string as it's input parameter, returns the Latitude value of the grid reference passed in the input parameter. If invalid, then the return value is blank and the error can be accessed using GetLastError
ConvertToLong	Function	WideString	Takes a string as it's input parameter, returns the Longitude value of the grid reference passed in the input parameter. If invalid, then the return value is blank and the error can be accessed using GetLastError
ValidSpatialRef	Function	WordBool	Takes a string parameter as the spatial reference and returns a WordBool value to indicate whether the passed parameter was deemed to be valid for the COM spatial reference system.
ConvertFrom LatLong	Function	WideString	Takes 2 input parameters, iLat and iLong. Returns the spatial reference equivalent of these values. If invalid, then the return value is blank and the error can be accessed using GetLastError

Formatted: Table Row

GetLastError	Readonly Property	WideString	Returns an error string after any translation failure.

Formatted: Table Row

IBaseMapFormat

This interface is implemented in addition to ISpatialReference by objects that allow base maps to be displayed in that projection. To allow this, functions are provided to convert easting/northing (the position of each point on the map in the coordinates used in the source map file) to Latitude and Longitude.

Attribute	Type	Data Type	Description
MapCoordToLat	Function	WideString	Takes 2 input parameters, the easting and northing read from the map. Returns a latitude.
MapCoordToLong	Function	WideString	Takes 2 input parameters, the easting and northing read from the map. Returns a longitude.
CanDisplayGrids	Readonly property	Boolean	Return true if the base map projection is not distorted, that is a fixed distance north or east has the same number of map pixels no matter where on the map it is measured. This allows Recorder 2000 to enable the grid lines and distribution point sizes options when viewing a map.
MapCoordsPer Metre	ReadOnly property	Double	Number of map coordinate points that make up a single metre, used for supplying correct grid scales and point sizes. For example, OSGB base map files have a system where a difference of 1 in the Easting is equal to 1 metre, so this value is 1.
LatLongToMapCoordX	Function	Double	Takes to input parameters, the Latitude and Longitude. Converts these to an Easting value appropriate to the base map's system.
LatLongToMapCoordY	Function	Double	Takes to input parameters, the Latitude and Longitude. Converts these to an Northing value appropriate to the base map's system.

Formatted: Table Row

IRecordCardHeader

This interface allows an ActiveX control / Active Form to replace the controls in the pnlHeader section of frmPlaceCard.

A record card may implement IRecorderFormEvents to gain access to existing controls on the Record Card, although the events are not triggered. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
SaveHeaderInfo	Function	WideString	Called from the Save button on frmPlaceCard. Saves data to the database, returns the SAMPLE_KEY of the sample the header information creates if successful and the blank string '' otherwise.
GetLastError	Readonly property	WideString	Returns an error string after any failure to save header info.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IAdditionalFilterDialog

This interface allows an ActiveX control / Active Form to replace the additional filter dialog, dlgComplexFilter. The class implementing IAdditionalFilterDialog must also implement IDialog in order that the size information can be specified. Finally, the IFormCaption interface may be optionally implemented. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
AddFilterableField	procedure	None	Takes a Widestring parameter which indicates the key value of a USABLE_FIELD record which can be filtered in the current query. Thie procedure is called repetitively to add all the possible filterable fields, finally FinishAddingFields is called.
FinishAddingFields	procedure	None	Indicates that all the possible filterable fields have now been sent to the dialog.
GetNextFilter	function	IAdditional Filter	Repetitively called by Recorder 2000 when the Ok button is clicked on the dialog, allowing the Addin to return all the Additional Filters the user has created. The Addin must return nil to indicate the last filter has been returned, and no more than 254 filters can be returned.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IAdditionalFilter

This interface allows IAdditionalFilterDialog addins to inform Recorder 2000 of the filters that must be applied. It is not an Addin in itself, so the class does not need to implement IRecorderAddin. Addins return multiple instances of this interface to Recorder 2000 when the dialog OK button is clicked.

Attribute	Type	Data Type	Description
FieldKey	Read only property	WideString	Indicates the USABLE_FIELD_KEY of a record in USABLE_FIELD on which the filter operates.
Criteria	Read only property	WideString	Actual data value being tested for. For example, '1/8/1998 and 1/9/1999' could be specified where Condition = 'is Between'.
Condition	Read only property	WideString	Indicates the test condition to be applied. Valid values are : is Equal to is Not Equal to Contains Starts with is Greater than is Less than is Between
AndOperation	Read only property	WordBool	Flag indicating if the filter is 'and'ed (true) or 'or'ed (false) to the previous filter.

Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: English (U.K.)
Formatted: Table Row, Widow/Orphan control
Formatted: Table Row

IRecorderFormEvents

This interface allows a class to register 'interest' in a particular existing form. The Form is passed to the class as an IRecorderForm interface, and various events occurring on the form trigger procedures in the class.

A single instance of the addin is created to handle IRecorderFormEvents, in isolation from other interfaces that are not directly associated with the form. Therefore, if an addin implements IRecorderFormEvents alongside INewAction or utilising ShowActiveForm, then multiple instances of the addin class are created. To avoid this problem, alter the COM Object Factory to allow singleton access to the addin.

The IRecorderForm interface in itself provides customisation facilities for the form.

Attribute	Type	Data Type	Description
FormName	Read only property	WideString	Allows the class to declare which form it is 'interested' in. Any form derived from TBaseForm may be used – the string returned must be a valid form class name as detailed in

Formatted: Table Row
Formatted: Table Row

			the TSD (eg TfrmLocations).
DoItemChange	Procedure	None	Triggered when the currently selected data item changes in the form. For example, navigation through a hierarchy triggers this procedure each time a node is clicked on. Takes a table name and key value as widestring parameters. The addin must not display a modal dialog in response to this event: use it to update the addin's state rather than to communicate with the user.
CheckCanSave	Function	Boolean	Triggered when the user attempts to save data on the form. Allows the addin to perform validation. Return false to prevent the form from saving in its current state.
DoSave	Procedure	None	Informs the addin that a save operation is taking place.
DoCancel	Procedure	None	Informs the addin that a cancel operation is taking place.
SetForm	Procedure	None	Passes an IRecorderForm interface as a parameter. This allows the addin to gain access to controls on the form.
DoEditmode	Procedure	None	Informs the addin that the form is going into edit mode.
DoAdd	Procedure	None	Informs the addin that a new record is to be added to the form.
DoDelete	Procedure	None	Informs the addin that the current record is to be deleted.

Formatted: Table Row

IProvidesOccurrencesSQL

Implement this interface in addin screens to allow the addin to provide SQL to Recorder that lists occurrences for the currently selected item. This allows the Quick Report menu item to operate within Recorder.

Attribute	Type	Data Type	Description
CanProvideSQL	Readonly property	Boolean	Return true if the screen is in a state where it is able to provide SQL that lists occurrences for the selected item. For example, return false if there is no current selected item.

Formatted: Table Row

OccurrencesSQL	Readonly property	Widestring	Return the SQL that lists occurrences for the selected item. SQL must return the following columns: OCCURRENCE_KEY string TYPE string ('T' or 'B' for Taxon or Biotope). The Atypes parameter determines if taxa or biotopes are being requested. Possible values for this parameter are T, B, or TB. It is recommended that the SQL returned only contains the types requested, otherwise blank rows will appear in reports.

Formatted: Table Row

IDisplayData

Addins which implement this interface are declaring their ability to display data in response to a request to IRecorder2000.DisplayData. The data is provided to the addin in the form of a keylist.

Attribute	Type	Data Type	Description
DisplayData	Procedure	None	Takes a IKeyList parameter. The addin displays the content of the keylist in response.
SupportsTable	Function	Boolean	Called by Recorder when checking if the addin supports displaying data for a particular table. Takes a parameter which is the table name being checked. The addin performs a case-insensitive test on the table name and returns true or false.

Formatted: Table Row

Formatted: Table Row

Formatted: Table Row

IMapDropFormat

The following interface is implemented (along with IMapPoint) by addins that enable Recorder to interpret non-standard data types when they are dropped onto a map. The addin converts the dropped data into a list of IMapPoints to be displayed by a map dataset.

Attribute	Type	Data Type	Description
SupportedTableCount	read only property	Integer	Count of the number of table types that this addin is able to convert into a map dataset.

Formatted: Font: Bold

Formatted: Table Row

SupportedTable	function	String	Takes an index parameter. Retrieves the name of the supported table at that index position.
InitialiseDataset	procedure	none	Takes an IKeyList parameter. This passes the list of records in a table that must be converted to a list of points to draw on the map. The table requested is previously checked by a call to SupportsInputTableName by Recorder.
DatasetTableName	read only property	string	After a call to InitialiseDataset, returns the table name that the points in the dataset are linked to.
MapPointCount	read only property	integer	Returns the number of distribution points generated from the key list passed after a call to InitialiseDataset.
MapPoint[]	read only property	IMapPoint	Returns the details of the distribution point at the supplied index position in the list generated by a call to InitialiseDataset. Used by Recorder to read the list of points from the addin and draw them on the map.

IMapPoint

The following interface allows addins that implement IMapDropFormat to return details of a single point within a map dataset.

Attribute	Type	Data Type	Description
Lat	read only property	double	Latitude (E) value for the data point to draw.
Long	read only property	double	Longitude (N) value for the data point to draw.
Accuracy	read only property	integer	Precision of the point, in metres. For example, points recorded against a 10 km grid square return 10,000. Records made against a point based grid system should return zero. Recorder uses this value to hide distribution points where the

Formatted: Font: Bold

Formatted: Table Row

			accuracy means the point may be misleading, for example a 1km disrtibution map does not show records made with 10km accuracy.
RecordKey	read only property	string	Primary key of the record being plotted. Combined with ImapDropFormat.OutputTableName this is used to locate the source record when the Find Source Data tool is used.
Date	read only property	datetime	Returns the date of the record. Used to draw hollow and closed distribution points for observations made before or after a cut off year.
Value	read only property	integer	Reserved for future use.

IMapWindowSelector

Any addin presenting a list of available base maps implements this interface so that it can be notified and is able to refresh anything to do with maps (e.g. the button on the spatial reference component).

Attribute	Type	Data Type	Description
UpdateMapWindowSelector	procedure	None	Called by Recorder whenever a configuration parameter is changed on the map, or map options dialog. The implemented method should use the additional properties of ICurrentSettings6 to get to the installed maps and the IAvailableMap interface to get the details of the base maps

Formatted: Table Row

Formatted: Table Row

IRecorderDetailScreen

Attribute	Type	Data Type	Description

Formatted: Font: Bold

Formatted: Table Row

EditRecord	Procedure	None	
DeleteRecord	Procedure	None	
Editing	Read-only property	Boolean	
Events	Read/Write proprerty	IRecorderDetailScreenEvents	
LoadContent	Procedure	None	Takes 3 string parameters: TypeName, ParentItemKey, ItemKey
SelectedItemCaption	Read-only property	String	
CanClose	Read-only property	Boolean	

Formatted: Table Row

IRecorderDetailScreenEvents

Attribute	Type	Data Type	Description
OnEditModeChange	Procedure	None	
OnRefreshScreenInfo	Procedure	None	Takes 2 string parameters: Key, Caption.

Formatted: Font: Bold

Formatted: Table Row

Menu Management Interfaces

The following interfaces allow addins to splice their own menu structure into Recorder's.

`IMergeMenuManager`

IMergeMenuManager allows addins to provide menus that apply to one specific MDI child (e.g. the Collections Browser). It is implemented by ActiveX addins that provide MDI screens embedded into Recorder.

Attribute	Type	Data Type	Description
DynamicMenuList	read only property	IDynamicMenuList	Returns the dynamic menu item list which is merged into the Recorder menus whilst the screen implementing IMergeMenuManager

Formatted: Font: Bold

Formatted: Table Row

			is active.

IDynamicMenuList

Implemented in one of 2 cases:

Implemented by addins that implement IMergeMenuManager allowing them to return details for the menu items to be merged into the main menu when the screen implementing IMergeMenuManager is active.

Implemented by addins that do not implement IMergeMenuManager, in this case the menu items are combined into the Recorder main menu throughout the lifetime of the application session.

Attribute	Type	Data Type	Description
Count	read only property	integer	Returns the number of menu items the addin wants to add to Recorder's main menu.
Items	read only property	IDynamicMenu	Takes an index property. Returns the IDynamicMenu object that describes the item at this position.
ImageListHandle	read only property	Handle	The Windows handle of the Image List used to provide images that appear in the menu alongside each item. If not used, then return 0.
MenuPath	read only property	string	Takes a parameter, AIndex, indicating the menu item in the list that is requested. Path to the menu item, separated by '\'. For example, 'Data Entry\New Data Entry Screen'.
InsertBeforeMenu	read only property	string	Takes a parameter, AIndex, indicating the menu item in the list that is requested. Path to the menu item that this menu item is inserted before, separated by '\'. For example, 'Data Entry\Observations'. Note that the addin should implement only one of InsertBeforeMenu and InsertAfterMenu for a given item. The unused method returns an empty string.
InsertAfterMenu	read only	string	Takes a parameter, AIndex, indicating the menu item in the list that is

Formatted: Font: Bold

Formatted: Table Row

	property		requested. Path to the menu item, separated by '\'. For example, 'Data Entry\Observations'.

IDynamicMenu

Returns details for a single menu item within a dynamic menu list.

IDynamicMenu Interface			
Attribute	**Type**	**Data Type**	**Description**
Execute	function	IUnknown	Called from Recorder when an addin's menu item corresponding to the IDynamicMenu instance is clicked. Returns IUnknown. If the returned object implements IDialog then the returned object is embedded onto a modal dialog. If the returned object implements IExecuteAction then the object is executed. Otherwise the object (which must be an ActiveX) is embedded onto an application MDI child window.
ChildCount	read only property	integer	Returns the number of child menu items that should appear in a sub menu under this menu item.
Child	read only property	IDynamicMenu	Takes an Index parameter, returns the sub-menu item identified by the index.
GetItemData	procedure	none	GetItemData is called on inserting the menu items. The ActionData structure is passed to the procedure as a Var parameter. ActionData is a record structure initialised with default values, and should be appropriately populated before returning.
HasSubMenu	read only property	Boolean	Returns true if the menu item has a sub menu, even if the sub menu currently contains no items.

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row

Options Dialog Interfaces

The following interfaces allow addins to register one or more pages on the Options dialog. The addin class implements IOptionsPages, which returns one or more ActiveX controls implementing the IOptionsPage interface.

IOptionsPages Interface			
Attribute	**Type**	**Data Type**	**Description**
Count	read only property	Integer	Returns the count of options pages that the addin provides.
Items	read only property	IOptionsPage	Takes an index parameter, returns the options page instance for that index.

IOptionsPage Interface			
Attribute	**Type**	**Data Type**	**Description**
Title	read only property	Widestring	Returns the title of the page.
Load	procedure	None	Instructs the page to load its data from the current persisted settings.
Save	procedure	None	Instructs the page to save its data.
Default	procedure	None	Instructs the page to reload its default settings.

Hierarchy Nodes Interfaces

The observations hierarchy is able to list and allow data entry for additional types of occurrences.

Additional occurrences nodes are able to contain child nodes for contained data items such as measurements and determinations.

Note that addins should not implement INodeManager, but they should implement IOccurrenceNodeManager instead. This a direct descendant of INodeManager with the same list of methods that require implementation. This approach also allows extensibility to support other hierarchy types in future.

INodeManager Interface

Attribute	Type	Data Type	Description
AddNode	Function	Boolean	Called when a request to add a node managed by the addin is made. TypeID parameter is the type of node to add. NodeInfo parameter should be updated with the necessary information for the node to be properly displayed in the tree. Set the result value to False to prevent the node being added.
DeleteNode	Function	Boolean	Called when a request to delete a node managed by the addin is made. NodeInfo parameter contains the information about the node about to be deleted. Set the result value to False to prevent the node being deleted.
EditNode	Function	Boolean	Called when a request to edit a node managed by the addin is made. NodeInfo parameter contains the current information about the node, and should be updated for the node to be properly refreshed. Set the result value to False to cancel any changes.
GetAddSubType	Function	Boolean	Called when updating menus for as long as there are items returned. This function lets the addin choose the items to appear in a separated section of the menu depending in the value of ParentTypeID. Any item returned is considered to correspond to a type belonging exclusively to the specified ParentTypeID. ParentTypeID parameter is the type associated with the node currenly selected in the tree. A value of -1

Formatted: Font: Bold

Formatted: Table Row

			indicates the selected node is entirely managed by the application. Any other value should match one type managed by the addin. Index parameter is incremented from 0 each time after this function is called. If the function returns True, the TypeID parameter should be set to the type to appear in the menus at the specified Index. The caption displayed in the menu will be retrieved by using this TypeID with the GetTypeName function. Set the result to False to indicate there are no more menu items for the specified ParentTypeID.
GetAddType	Function	Boolean	Called when updating menus for as long as there are items returned. This function lets the addin choose the items that will be appearing on the menus depending on the value of ParentTypeID. ParentTypeID parameter is the type associated with the node currenly selected in the tree. A value of -1 indicates the selected node is entirely managed by the application. Any other value should match one type managed by the addin. Index parameter is incremented from 0 each time after this function is called. If the function returns True, the TypeID parameter should be set to the type to appear in the menus at the specified Index. The caption displayed in the menu will be retrieved by using this TypeID with the GetTypeName function. Set the result to False to indicate there are no more menu items for the specified ParentTypeID.

GetData	Function	Boolean	Called when a node is expanded. This function lets the addin set the SQL statement that will be used to query the database and populate the target tree. ParentTypeID parameter is the type associated with the parent node of the currenly selected one in the tree. A value of -1 indicates the selected node is entirely managed by the application. Any other value should match one type managed by the addin. TypeID parameter is the type of node for which the data is requested. ParentKey parameter is the database key of the parent node. SQL parameter should be set to a valid SQL statement that will be run against the database. Set the result to False to indicate nothing to do.
GetNodeInfo	Function	Boolean	Called for each record in the dataset resulting from querying the database with the SQL statement provided by GetData. TypeID parameter is the type of the nodes being added. DataFields parameter is the current record being processed, and can be used to set the properties of the node, using NodeInfo. NodeInfo parameter is initialised to default values on entry. Set the result value to False to prevent a node being added to the tree.
GetSubNodeInfo	Function	Boolean	Called for each node returned by GetNodeInfo. This function lets the addin choose the types to appear below the specified

			ParentTypeID. These can be as many or as few as required. ParentTypeID parameter is the type associated with the node for which the items are requested. Index parameter is incremented from 0 each time after this function is called. NodeInfo parameter is initialised to default values and should be updated for the sub-nodes to be displayed properly. Set the result value to False to prevent a node being added to the tree.
NodeChanged	Function	Boolean	Called after a node managed by the addin becomes the current node. The ParentItemKey parameter is the key associated with the parent node. The NodeInfo parameter contains the current information about the node, and can be changed for the node to be properly updated in the tree. The Editable/Deletable state should be updated here and will be reflected in the main application on return. The DetailScreenGUID parameter is the GUID of the activeX class that will be instanciated and displayed by Recorder. The ActiveX class must implement IRecorderDetailScreen. If the node doesn't have an associated detail screen, do not change its initial value, set to '{00000000-0000-0000-0000-000000000000}'. Set the result value False if there is no detail screen to display.
ImageList	Read only property	Handle (longword)	Returns the handle of the imagelist holding the node images.
StateImageList	Read only property	Handle (longword)	Returns the handle of the imagelist holding the treeview state images.

Formatted: English (U.K.)

TypeCount	Read only property	Handle (longword)	Returns the number of different node types supported by the addin.
TypeID	Read only property	Handle (longword)	Takes an index parameter. Returns the proper TypeID of each supported type, as specified by GetTypeCount. -1 is a reserved value and should not be used.
TypeName	Read only property	WideString	Take a TypeID parameter. Returns the name of the node identified by the specified TypeID parameter.

Formatted: English (U.K.)

Grid System Interface

This interface is implemented by addins in addition to ISpatialRefence in order to declare that the spatial reference system has additional functionality related to grid squares.

IGridSystem Interface			
Attribute	**Type**	**Data Type**	**Description**
InitialiseGridSquareRange	Procedure	None	Passes the Southwest and Northeast corners of range of grid squares. Prepares a list of all the grid squares in the box defined by these 2 corners. The list is subsequently accessesd by calling GridSquareRangeCount and GridSquareRangeItem.
GridSquareRangeCount	Read only property	Integer	Returns the number of grid squares previously identified by a call to InitialiseGridSquareRange.
GridSquareRangeItem	Read only property	WideString	Takes an index as a parameter. Returns the grid square for this index as previously identified by a call to InitialiseGridSquareRange.
ReduceGridSquarePrecision	Function	WideString	Takes a grid square and the precision (metres), returns a grid square with the precision reduced to the square of the size specified by the precision. For example,

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row

			ReduceGridSquarePrecision('ST789123', 10000) returns 'ST71'.
GridSquareAsENBox	Function	ILatLongBox	Takes a grid square as a parameter. Returns a box defining the Southwest and Northeast corners of the box. The box returned must use the same system as the base map system the grid operates on (identified by ISpatialReference6.EquivalentBaseMapSystem). This is a coordinate based easting northing system as used by the underlying map.

Formatted: Normal

Easting Northing Box Interface

Datatype used to transfer a bounding box defined in easting and northing coordinates.

IENBox Interface			
Attribute	**Type**	**Data Type**	**Description**
SWE	read only property	double	Easting value for the southwest corner of the bounding box.
SWN	read only property	double	Northing value for the southwest corner of the bounding box.
NEE	read only property	double	Easting value for the northeast corner of the bounding box.
NEN	read only property	double	Northing value for the northeast corner of the bounding box.
System	read only property	WideString	Identifies the spatial reference system.

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row
Formatted: Normal

Spatial Reference List Interface

Allows a single addin to declare support for multiple spatial reference systems. Addins implementing ISpatialReferenceList do not implement ISpatialReference themselves, but instantiate and return additional COM objects which implement ISpatialReference separately.

ISpatialReferenceList Interface

Formatted: Font: Bold
Formatted: Table Row, Centered

Attribute	Type	Data Type	Description
SystemCount	Read only property	Integer	Number of spatial reference systems supported by this addin.
SystemInterface	Read only property (indexed)	ISpatialReference	Called by Recorder to access a specific spatial reference system in the list. The addin returns a COM object implementing ISpatialReference.

Formatted: Font: Bold
Formatted: Table Row
Formatted: Normal

Base Map Format List Interface

Allows a single addin to declare support for multiple base map format systems. Addins implementing IBaseMapFormatList do not implement IBaseMapFormat themselves, but instantiate and return additional COM objects which implement IBaseMapFormat separately.

IBaseMapFormatList Interface			
Attribute	**Type**	**Data Type**	**Description**
SystemCount	Read only property	Integer	Number of base map format systems supported by this addin.
SystemInterface	Read only property (indexed)	IBaseMapFormat	Called by Recorder to access a specific base map format system in the list. The addin returns a COM object implementing IBaseMapFormat.

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row
Formatted: introduction

Report Keytype List Interface

Allows an addin to declare any number of additional keytypes that should be supported by the Xml Reports facility. For example, an addin that provides a facility to store data in a new entity called Collection can register the Collection keytype so that it is possible to have Xml Reports that filter on the Collection primary key attribute.

IReportKeytypeList Interface			
Attribute	**Type**	**Data Type**	**Description**
KeytypeCount	Read only property	Integer	Number of keytypes registered by this addin.
Keytype	Read only property	IKeytype	Provides access to details of each

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row

	(indexed)		keytype registered by the addin.

Formatted: Normal

Report Keytype Interface

Allows an addin to declare details of a single keytype being registered for use by the Xml Reports facility..

IReportKeytype Interface			
Attribute	**Type**	**Data Type**	**Description**
Name	Read only property	string	Name of the keytype, which must match the keytype attribute in the Xml Reports that are being made available by this registration.
Multivalue	Read only property	Boolean	Set to true if the keytype allows a list of keys to be used as a parameter instead of just a single key value. When set to true, the SQL filter in the Xml Report Where clause has the form: WHERE Keyfield IN (%s) In this case the %s is replaced by a comma separated list of the key values, with each item in quotes. If set to false, the Xml Report Where clause has the form: WHERE Keyfield = '%s' In this case the single parameter does not have quotes wrapping it unless specified in the SQL.

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row
Formatted: introduction

Named Spatial System Interface

For individual implementations of ISpatialReference, the display caption for the system is obtained from the addin name (e.g. displayed on the Tools\Options dialog). This is not available when implementing ISpatialReferenceList as all systems have the same addin name. Implemented alongside ISpatialReference, this provides a means of specifying the caption.

INamedSpatialSystem Interface			
Attribute	**Type**	**Data Type**	**Description**
Caption	Read only property	String	Display caption for the system.

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row

Base Map Scale List Interface

Implemented alongside IBaseMapFormat, this interface allows a base map format to additionally specify the sizes and captions used for grid square sizes (for the grid lines menu) and distribution point sizes (for the distribution point sizes menu). If not implemented then the default settings are used.

IBaseMapScaleList Interface			
Attribute	**Type**	**Data Type**	**Description**
GridScaleCount	Read only property	Integer	Number of grid scales supported in the grid lines menu.
GridScaleCaption	Read only property (indexed)	String	Caption for the scale in the list specified by the index parameter.
GridScaleX	Read only property (indexed)	Double	Size of the X dimension of the grid square in the list specified by the index parameter. Units are the same as the units used in the underlying base map file format.
GridScaleY	Read only property (indexed)	Double	Size of the Y dimension of the grid square in the list specified by the index parameter. Units are the same as the units used in the underlying base map file format.
PointSizeCount	Read only property (indexed)	Integer	Number of point sizes supported in the grid lines menu.
PointSizeCaption	Read only property (indexed)	String	Caption for the point size in the list specified by the index parameter.
PointSizeX	Read only property	Double	Size of the X dimension of the point size in the list specified by the index

Formatted: Font: Bold
Formatted: Table Row, Centered
Formatted: Font: Bold
Formatted: Table Row

	(indexed)		parameter. Units are the same as the units used in the underlying base map file format.
PointSizeY	Read only property (indexed)	Double	Size of the Y dimension of the point size in the list specified by the index parameter. Units are the same as the units used in the underlying base map file format.

Formatted: Normal

Base Map Scale List 614 Interface

Implemented alongside, and as an extension of, IBaseMapScaleList, this interface allows to additionally specify the amounts to offset the grid lines by. If not implemented, the default settings are used (0 for both values).

IBaseMapScaleList_614 Interface			
Attribute	**Type**	**Data Type**	**Description**
GridOffsetX	Read only property (indexed)	Double	Amount to offset the grid lines by in the X dimension in the list specified by the index parameter. Each item in the list corresponds to the similarly indexed GridScaleX. Units are the same as the units used in the underlying base map file format.
GridOffsetY	Read only property (indexed)	Double	Amount to offset the grid lines by in the Y dimension in the list specified by the index parameter. Each item in the list corresponds to the similarly indexed GridScaleY. Units are the same as the units used in the underlying base map file format.

Formatted: Normal

Recorder 2000 and Addin shared interfaces

The following interfaces may be implemented by both addins and Recorder 2000. They allow the transfer of lists of data items between the client and server.

IKeyList

This interface allows a COM object to extract data from key lists (defined in TSD) that are passed to it. Used primarily by COM objects supporting the IExportFilter and

IMapExportFilter interfaces. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
ItemCount	Read only Property	Integer	The number of items contained within the key list.
TableName	Read only Property	WideString	The name of the table the key list contains keys from. In the case of key lists containing both Biotope and Taxon occurrences, this will be set to 'MIXED'. KeyField2 of IKeyItem is then used to populate the table name of the particular occurrence record. See Below.
GetKeyItem	Function	IKeyItem	Retrieves a key item from the list according to the index parameter of the function.

Formatted: Table Row

IKeyItem

Used in conjunction with IKeyList, this interface defines the items on an IKeyList. The following table illustrates the properties and methods contained in this interface:

Attribute	Type	Data Type	Description
KeyField1	Read only Property	WideString	Contains the 16 character unique ID of the key item.
KeyField2	Read only Property	WideString	Used when the TableName property of the IKeyList to which the IKeyItem belongs is "MIXED". Specifies the table to which the key item applies.

Formatted: Table Row

ILatLong

Simple interface used only for passing latitude and longitude data between Recorder 2000 and com addins.

Attribute	Type	Data Type	Description
Latitude	Readonly Property	double	Identifies the latitude.
Longitude	Readonly property	double	Identifies the longitude
ErrorMsg	Readonly property	Widestring	Where Recorder 2000 is unable to return a valid latitude and longitude for any reason, the

Formatted: English (U.K.)
Formatted: Table Row

			ErrorMsg field identifies the reason.

IVagueDate

Simple interface used only for vague dates between Recorder 2000 and com addins.

Vague dates are constructs of 3 fields which allow an imprecise date to be recorded and queried in Recorder 2000. The principal is that 2 date fields identify the range of dates which are encompassed by the vague date, and a third field specifies the type of date. This allows concepts such as Spring 2000, before 1979, or exact dates to be recorded.

Attribute	Type	Data Type	Description
StartDate	Readonly property	TDateTime	Start of the possible date range for the vague date.
EndDate	Readonly property	TDateTime	End of the possible date range for the vague date.
DateType	Readonly property	WideString	Type of vague date
ErrorMsg	Readonly property	WideString	If a vague date is attempted from a string that cannot be recognised as a valid date, then the Type property is set to 'Error' and the ErrorMsg property contains text identifying the reason. Otherwise this property is ignored.

Formatted: English (U.K.)
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

The following table illustrates each possible type of vague date:

Type	Example Date string	StartDate	EndDate
D	23 MAR 1987	23 MAR 1987	23 MAR 1987
DD	23 MAR 1987 - 30 MAR 1987	23 MAR 1987	30 MAR 1987
O	MAR 1987	01 MAR 1987	31 MAR 1987
OO	MAR 1987 - MAY 1987	01 MAR 1987	31 MAY 1987
P	SUMMER 1987	01 JUN 1987	31 AUG 1987
Y	1987	01 JAN 1987	31 DEC 1987
YY	1981-1983	01 JAN 1981	31 DEC 1983
Y-	1980-	01 JAN 1980	<ignore>

Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

-Y	-1979	<ignore>	31 DEC 1979
M	July	01 JUL 9999	31 JUL 9999
S	Autumn	01 SEP 9999	30 OCT 9999
U	Unknown	<ignore>	<ignore>

Formatted: English (U.K.)
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row
Formatted: Table Row

Clipboard and Drag/Drop implementation

In order that Addins can drag and drop data onto existing recorder forms, they must provide data in the CF_JNCCDATA format. This format's registration number is supplied to all addins via the IRecorder2000 interface. Addins must provide standard Windows IDropTarget classes when implementing drag and drop into the main Recorder 2000 application. Instructions on implementing standard Drag and Drop interfaces are available in the Win32 SDK.

CF_JNCCDATA Clipboard Format

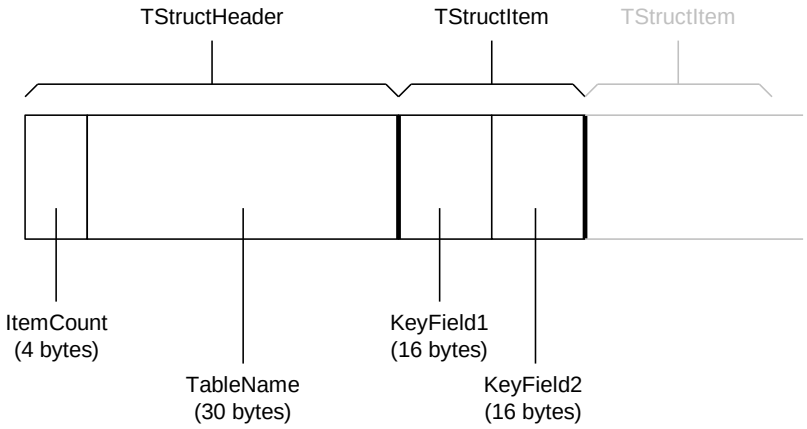
The CF_JNCCDATA clipboard format always begins with a structure of type TDropStructHeader.

TDropStructHeader Fields	
ItemCount : Integer	The number of data items in the clipboard structure.
TableName : string[30]	The name of the database entity which is being dragged.

Following the structure are any number of TDropStructItem records (identified by the ItemCount).

TDropStructItem Fields	
KeyField1 : TKeyString	The primary key field for the data item within the table.
KeyField2 : TKeyString	If the primary key is compound, then specifies the second field's key value, otherwise blank.

The following diagram illustrates the memory structure of data dropped in CF_JNCCDATA format:



The same CF_JNCCDATA format is used to share data via the clipboard.

Installing COM objects

Installation of COM addins into Recorder 2000 is fully automated within the Install Addins dialog. The dialog requires a valid OCX or DLL containing addins to install, and any bitmaps specified by the addins must be in the same folder.

The installation procedure creates the following registry entries for Recorder 2000.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\JNCC\Recorder\Installed Addins\ ServerName.ClassName]
"Installed"=1
"ClsId"= The class ID of the COM object
```

where **ServerName** is the name of the DLL / EXE / OCX file and **ClassName** is the name of the class within that file.

In addition, the library is registered with the operating system in the standard fashion as specified by Microsoft.

Creating NBN keys

When new data is added to the database from within a COM addin, it is essential that the key values for each record are generated correcly.

Each key value consists of a 16 character string divided into 2 eight character sections. The first eight characters are the site copy identifier and can be obtained from the registry setting HKEY_CURRENT_USER\Software\JNCC\Recorder\Settings\Site ID.

The second eight characters are an incrementing key value for the record, used to ensure that the records all have a unique identifier on each database copy. As each database copy has its own site id, every key value created within the NBN is unique for

each table. To generate a new set of characters for a new record, the last used string can be obtained for the table from the LAST_KEY table. This set of characters is incremented according to the rules following:

Increment the rightmost character within the character range 0..9 and A..Z (so that 9 increments to A etc).

If this character exceeds the given range (ie exceeds Z), then set it to 0 and increment the next character on the left. If this also exceeds Z then also set it to 0 and increment the next character on the left, until all characters are used.

Therefore, the lowest key value is 00000000 and the highest is ZZZZZZZZ. This gives nearly 3 billion combinations (ie 3 thousand million).

Extending the data model using Addins

The data model may be extended through installed addins. To achieve this, certain rules must be followed.

Data model modifications should not take the form of alterations to existing data entities. Addition of new fields to existing data may be achieved by creating a table with a 'one to one' join to an existing table. For example, addition of a 'benthic/pelagic' flag for occurrences recorded using a marine addin would require a new table, MARINE_OCCURRENCES, with a TAXON_OCCURRENCE_ID field to join it to the existing TAXON_OCCURRENCE table.

Extending the Document Type Definition

In order that the additional data is to be exported and imported correctly, it is necessary to extend the DTD, which defines the structure of XML output to NBN standards.

Where field or table names do not match those specified in additional DTD sections, it is necessary to add records to the SPECIAL_XML_ELEMENT table to specify the mappings. In all cases, the NAME field represents the name of the element defined. The TYPE field can have one of three values, 'F', 'T', or 'M'. The DATA field indicates the field or table name after translation. Further information is given in the table definition below.

SPECIAL_XML_ELEMENT Table Definition

This table defines elements which are output in NBN export files, but do not relate to a specific database field or table. For example, the metadata elements which are output with every XML file are defined here. Further information is given in the section on XML export.

COLUMN NAME	DESCRIPTION	TYPE	SIZE
NAME	Unique identifier for the SPECIAL_XML_ELEMENTS table. Name of the element which has fixed output text defined.	Text	50
TYPE	Identifies the type of element. 'M' = metadata element. Metadata elements are used to provide data such as dataset_source information in the metadata element. The DATA field provides the text for tags with a matching NAME. 'F' = field translation, 'T' = table translation. These types allow fields or tables to have different names in the Recorder data model to those specified in the DTD. The NAME field is the name of the DTD element, and the DATA field is a semi-colon separated list of possible field/table names. These are combined with a specifier where a specifier attribute is present. The DATA semi-colon list is in order or preference.	Text	1
DATA	For metadata elements, identifies the text to be placed inside the tag. Some elements are set up as part of the installation procedure: For T and F translation elements, indicates the name of the field or table to be used where an element of name NAME is encountered. Semi-colon separated lists can be used where several options are possible – order of precedence is as presented in the list. For field translations, a '*' value indicates that the table's primary key should be used.	Memo	

CREATING COM OBJECTS IN DELPHI 4

Introduction

Delphi 4 has several objects and wizards to make the creation of COM objects, Automation objects and Active Forms a relatively straight-forward process. ActiveX controls will NOT be discussed in this document (not for Delphi 4 anyway, in microsoft's Visual Basic they use ActiveX control to mean an Active Form in Delphi). It is advisable to firstly create an ActiveX library (DLL) to house any of these objects, and it is this DLL that acts as a server for the objects it contains. Note it is possible to create just COM objects without including them in an ActiveX library.

Defining a type library

1. From Delphi 4 with no applications / files open, select **File / New / ActiveX / Type Library**. Click the **[OK]** button. *A new type library unit is created with a GUID created for it.*

2. Save the file using **File / Save** and selecting the correct location and name for the unit file. For example, C:\Temp\MyTypeLibrary.tlb.

3. Use the type library editor to add interfaces, dispinterfaces, properties and methods.

For Recorder 2000, a type library and corresponding *.pas file have been created which contain all supported interfaces, dispinterfaces, properties and methods for the application. This type library can then be used by Delphi, Visual Basic and any other development environment that supports the use of type libraries to access these interfaces etc. for the creation of COM objects etc.

The type library file for Recorder 2000 is Recorder2000.tlb and the corresponding Delphi unit file Recorder2000_TLB.pas.

Creating an ActiveX library

1. From Delphi 4 with no applications / files open, select **File / New / ActiveX / ActiveX Library**. Click the **[OK]** button. A new library unit is created which exports the required functions: DLLGetClassObject, DLLCanUnloadNow, DLLRegisterServer, DLLUNregisterServer.

2. Save the project using **File / Save** and selecting the correct location and name for the project file. For example, C:\Temp\MyServer.dpr.

We have now created an ActiveX server library which when built will create a DLL file of the same name as our project. Before we register our ActiveX server we need to create COM objects/Automation objects/Active Forms to put “inside” it.

Creating COM objects

1. With the ActiveX server library created above still open, select **File / New / ActiveX / Automation Object**. Click the **[OK]** button. *The COM Object Wizard is displayed prompting for entry of a Class Name for our new COM object.*

2. Enter a class name, description and any interfaces that the object will implement and click the **[OK]** button. *A new unit is created for the COM object with the type declaration something like :*

   TMyCOMObject = class(TComObject, Interface 1,...,Interface N)

3. Save the unit using **File / Save** and selecting the correct location and name for the unit file. For example, C:\Temp\MyCOMObject.pas.

4. In the following example, have created an ActiveX server library, DMAPServer.dll, a COM object, Exporter.pas which implements the IMapExportFilter of Recorder2000.tlb.

```
unit Exporter;

interface

uses
  SysUtils, Windows, ActiveX, ComObj, Recorder2000_TLB, Classes, accmain;

type
  EExporterError = class(Exception);

  TListType = (ltMixed, ltTaxon, ltBiotope, ltLocation, ltName, ltSurvey,
               ltEvent, ltSample);

  TCommaSeparatedKeys = record  //Deals with mixed lists
    Keys1 : string;  //for mixed lists these will be biotope occurrence keys.
    Keys2 : string;  //for mixed lists these will be taxon occurrence keys.
    ListType : TListType;
  end;

  TExporter = class(TAutoObject, IRecorderAddIn, IFilter, IExportFilter,
   IExportByColumn)
  private
    { Internal field variable to receive current settings from Recorder 2000 }
    FRecorder2000 : IRecorder2000;
    FMeasurementUnitKey : WideString;
    FNumberOfRanges : integer;
    FMinValue : Double;
    FMaxValue : Double;
    FTypeErrors : Boolean;
    FdbMain : TtaDatabase;
    FssMain : TtaSession;

    procedure OpenDatabase;
    procedure CloseDatabase;
    function BuildCommaSeparatedList(const iItemsToExport : IKeyList) :
   TCommaSeparatedKeys;
    function BuildSelect(const iBiotope : boolean) : string;
    function BuildFrom(const iListType : TListType; iBiotope : boolean) : string;
    function BuildPref(const iBiotope : boolean) : string;
    function BuildWhere(const iListType : TListType; const iKeys : string;
             const iBiotope : Boolean) : string;
    function BuildMeasurement(const iBiotope : boolean) : string;
    function BuildSort(const iBiotope : boolean) : string;

    procedure ExportOccurrences(const iBiotope: boolean;
              const iListType : TListType; const iKeys : string;
              var ioNAMFile, ioDISFile : TStringList; var ioIndex : integer);
    function TrimFirstWord(const iString : string) : string;
    function AddLeadingBrackets(const iString : string) : string;
    function RemoveSpaces(const iString : string) : string;
    function SetRangeValue(const iValue : double) : string;
    function RemoveExistingExt(const iFile : string) : string;
  protected
    { Recorder 2000 AddIn interface }
    function Get_Name: WideString; safecall;
    function Get_Description: WideString; safecall;
    function Get_ImageFileName: WideString; safecall;
    procedure SetRecorder2000Interface(const iRec2000Intf: IRecorder2000); safecall;
    procedure Install; safecall;
    { IFilter }
    function Get_DefaultFileExtension: WideString; safecall;
    { IExportFilter }
```

```
    function ExportFile(const iItemsToExport: IKeyList; const iDestinationFile:
   WideString): WordBool; safecall;
    function Get_LastExportError: WideString; safecall;
    { IExportByColumn }
    function Get_MeasurementUnitKey: WideString; safecall;
    procedure Set_MeasurementUnitKey(const Value: WideString); safecall;
    function Get_NumberOfRanges: Integer; safecall;
    procedure Set_NumberOfRanges(Value: Integer); safecall;
    function Get_MinValue: Double; safecall;
    procedure Set_MinValue(Value: Double); safecall;
    function Get_MaxValue: Double; safecall;
    procedure Set_MaxValue(Value: Double); safecall;
  public
  end;

const
  Class_Exporter: TGUID = '{F487D5E1-0DC4-11D3-B6DA-0060979160B0}';

implementation

uses ComServ, db, bsemain, taScrollStuff, JNCCDatasets,
     Dialogs, DataClasses, Registry;

const
  DIS_FILE_EXT='.dis';
  NAM_FILE_EXT='.nam';

  JNCC_SESSION = 'JNCCSession';
  JNCC_DATABASE = 'JNCCDatabase';

  EST_DBPATH_MISSING = 'Database Path not in registry - export cannot be performed';
  EST_PASSWORD_MISSING = ' Database password not in the registry - export cannot be
   performed';
  EST_WRONG_PASSWORD = 'The database password is incorrect';
  EST_ADD_VALUE = 'An error occurred writing additional column data.';
  EST_OPEN_FAIL = 'An error occurred trying to open the database.';

  FROM = 'From ';
  BIOTOPE_SELECT = 'SELECT BIOTOPE.BIOTOPE_KEY AS KEY_FIELD, BIOTOPE.SHORT_TERM AS
   NAME_FIELD, ' +
                   'SAMPLE.SPATIAL_REF, SAMPLE.LAT, SAMPLE.LONG, ' +
                   'SAMPLE.VAGUE_DATE_START, SAMPLE.VAGUE_DATE_END,
   SAMPLE.VAGUE_DATE_TYPE';
  TAXON_SELECT =   'SELECT TAXON.TAXON_KEY AS KEY_FIELD, TAXON.ITEM_NAME AS
   NAME_FIELD, ' +
                   'SAMPLE.SPATIAL_REF, SAMPLE.LAT, SAMPLE.LONG, ' +
                   'SAMPLE.VAGUE_DATE_START, SAMPLE.VAGUE_DATE_END,
   SAMPLE.VAGUE_DATE_TYPE';
  BIO_MEAS_SELECT =', BIOTOPE_OCCURRENCE_DATA.DATA';
  TAX_MEAS_SELECT =', TAXON_OCCURRENCE_DATA.DATA';
  BIOTOPE_SORT =   'ORDER BY BIOTOPE.SHORT_TERM, SAMPLE.SPATIAL_REF';
  TAXON_SORT =     'ORDER BY TAXON.ITEM_NAME, SAMPLE.SPATIAL_REF';
  BIOTOPE_WHERE =  'BIOTOPE_OCCURRENCE.BIOTOPE_OCCURRENCE_KEY IN ';
  TAXON_WHERE =    'TAXON_OCCURRENCE.TAXON_OCCURRENCE_KEY IN ';
  LOCATION_WHERE = 'SAMPLE.LOCATION_KEY IN ';
  BIO_DET_WHERE =  'BIOTOPE_DETERMINATION.DETERMINER IN ';
  TAX_DET_WHERE =  'TAXON_DETERMINATION.DETERMINER IN ';
  SURVEY_WHERE =   'SURVEY.SURVEY_KEY IN ';
  EVENT_WHERE =    'SURVEY_EVENT.SURVEY_EVENT_KEY IN ';
  SAMPLE_WHERE =   'SAMPLE.SAMPLE_KEY IN ';
  TAX_OCC_JOIN =   'SAMPLE INNER JOIN TAXON_OCCURRENCE ON ' +
                   'SAMPLE.SAMPLE_KEY = TAXON_OCCURRENCE.SAMPLE_KEY) INNER JOIN ' +
```

```
                  'TAXON_DETERMINATION ON TAXON_OCCURRENCE.TAXON_OCCURRENCE_KEY =
   TAXON_DETERMINATION.TAXON_OCCURRENCE_KEY) INNER JOIN ' +
                  'TAXON_LIST_ITEM ON TAXON_LIST_ITEM.TAXON_LIST_ITEM_KEY =
   TAXON_DETERMINATION.TAXON_LIST_ITEM_KEY) INNER JOIN ' +
                  'TAXON_VERSION ON TAXON_VERSION.TAXON_VERSION_KEY =
   TAXON_LIST_ITEM.TAXON_VERSION_KEY) INNER JOIN ' +
                  'TAXON ON TAXON.TAXON_KEY = TAXON_VERSION.TAXON_KEY';
  BIO_OCC_JOIN =   'SAMPLE INNER JOIN BIOTOPE_OCCURRENCE ON ' +
                  'SAMPLE.SAMPLE_KEY = BIOTOPE_OCCURRENCE.SAMPLE_KEY) INNER JOIN ' +
                  'BIOTOPE_DETERMINATION ON BIOTOPE_OCCURRENCE.BIOTOPE_OCCURRENCE_KEY
   = BIOTOPE_DETERMINATION.BIOTOPE_OCCURRENCE_KEY) INNER JOIN ' +
                  'BIOTOPE_LIST_ITEM ON BIOTOPE_LIST_ITEM.BIOTOPE_LIST_ITEM_KEY =
   BIOTOPE_DETERMINATION.BIOTOPE_LIST_ITEM_KEY) INNER JOIN ' +
                  'BIOTOPE ON BIOTOPE.BIOTOPE_KEY = BIOTOPE_LIST_ITEM.BIOTOPE_KEY';
  TAX_DATA_JOIN =  ') INNER JOIN TAXON_OCCURRENCE_DATA ON
   TAXON_OCCURRENCE.TAXON_OCCURRENCE_KEY =
   TAXON_OCCURRENCE_DATA.TAXON_OCCURRENCE_KEY';
  BIO_DATA_JOIN =  ') INNER JOIN BIOTOPE_OCCURRENCE_DATA ON
   BIOTOPE_OCCURRENCE.BIOTOPE_OCCURRENCE_KEY =
   BIOTOPE_OCCURRENCE_DATA.BIOTOPE_OCCURRENCE_KEY';
  EVENT_JOIN =     'SURVEY_EVENT INNER JOIN SAMPLE ON SURVEY_EVENT.SURVEY_EVENT_KEY =
   SAMPLE.SURVEY_EVENT_KEY) ';
  SURVEY_JOIN =    'SURVEY INNER JOIN SURVEY_EVENT ON SURVEY.SURVEY_KEY =
   SURVEY_EVENT.SURVEY_KEY) ';
  BIO_PREF =       'WHERE BIOTOPE_DETERMINATION.PREFERRED = -1 AND ';
  TAX_PREF =       'WHERE TAXON_DETERMINATION.PREFERRED = -1 AND ';
  TAX_MEASUREMENT =' AND TAXON_OCCURRENCE_DATA.MEASUREMENT_UNIT_KEY = ';
  BIO_MEASUREMENT =' AND BIOTOPE_OCCURRENCE_DATA.MEASUREMENT_UNIT_KEY = ';
{ TExporter }

//==============================================================================

//counts number of closing brackets and adds the same number of opening brackets
//to the front.
function TExporter.AddLeadingBrackets(const iString: string): string;
var
  lPos, lStringLen, lCount, lIndex : integer;
  lSearchString, lResult : string;
begin
  lResult := '';
  lCount := 0;
  lSearchString := iString;
  lStringLen := Length(lSearchString);
  while lStringLen > 0 do begin
    lPos := Pos(')', lSearchString);
    if lPos > 0 then begin
      lCount := lCount + 1;
      lSearchString := Copy(lSearchString, lPos + 1, lStringLen - lPos);
      lStringLen := Length(lSearchString);
    end else
      lStringLen := 0;
  end;
  for lIndex := 1 to lCount do
    lresult := lresult + '(';
  Result := lResult + iString;
end;

//==============================================================================
function TExporter.BuildCommaSeparatedList(
  const iItemsToExport: IKeyList): TCommaSeparatedKeys;
var
  ltfLookAtKeyField2 : boolean;
  lCurrentKeyItem : IKeyItem;
```

```
  lTableName : string;
  lIndex : integer;
  lListType : TListType;
begin
  Result.Keys1 := '';
  Result.Keys2 := '';
  ltfLookAtKeyField2 := False;
  lTableName := iItemsToExport.Get_TableName;
  if CompareText(lTableName, MIXED_DATA) = 0 then begin //dealing with biotope and
   Taxon occurrences
    ltfLookAtKeyField2 := True;
    lListType := ltMixed;
  end;

  if CompareText(lTableName, 'Taxon_Occurrence') = 0 then
    lListType := ltTaxon;

  if CompareText(lTableName, 'Biotope_Occurrence') = 0 then
    lListType := ltBiotope;

  if CompareText(lTableName, 'Location') = 0 then
    lListType := ltLocation;

  if CompareText(lTableName, 'Name') = 0 then
    lListType := ltName;

  if CompareText(lTableName, 'Survey') = 0  then
    lListType := ltSurvey;

  if CompareText(lTableName, 'Survey_Event') = 0 then
    lListType := ltEvent;

  if CompareText(lTableName, 'Sample') = 0 then
    lListType := ltSample;

  for lIndex := 0 to iItemsToExport.Get_ItemCount - 1 do begin
    lCurrentKeyItem := iItemsToExport.GetKeyItem(lIndex);
    if ltfLookAtKeyField2 then begin
      if CompareText(lCurrentKeyItem.KeyField2, 'Biotope_Occurrence') = 0 then begin
        if Result.Keys1 <> '' then
          Result.Keys1 := Result.Keys1 + ', ';
        Result.Keys1 := Result.Keys1 + '"' + lCurrentKeyItem.KeyField1 + '"';
      end else if CompareText(lCurrentKeyItem.KeyField2, 'Taxon_Occurrence') = 0 then
   begin
        if Result.Keys2 <> '' then
          Result.Keys2 := Result.Keys2 + ', ';
        Result.Keys2 := Result.Keys2 + '"' + lCurrentKeyItem.KeyField1 + '"';
      end;
    end else begin
      if Result.Keys1 <> '' then
        Result.Keys1 := Result.Keys1 + ', ';
      Result.Keys1 := Result.Keys1 + '"' + lCurrentKeyItem.KeyField1 + '"';
    end;
  end;
  Result.ListType := lListType;
end;

//=============================================================================
function TExporter.BuildFrom(const iListType: TListType; iBiotope : boolean): string;
var
  lResult : string;
begin
  lresult := '';
```

```
  case iListType of
  ltEvent:
    begin
    lresult := EVENT_JOIN;
    end;
  ltSurvey:
    begin
    lresult := SURVEY_JOIN;
    lresult := lResult + TrimFirstWord(EVENT_JOIN);
    end;
  end;
  if iBiotope then begin
    if lresult <> '' then
      lresult := lresult + TrimFirstWord(BIO_OCC_JOIN)
    else
      lresult := lresult + BIO_OCC_JOIN;
  end else begin
    if lresult <> '' then
      lresult := lresult + TrimFirstWord(TAX_OCC_JOIN)
    else
      lresult := lresult + TAX_OCC_JOIN;
  end;
  if FMeasurementUnitKey <> '' then begin
    if iBiotope then
      lResult := lResult + BIO_DATA_JOIN
    else
      lResult := lResult + TAX_DATA_JOIN;
  end;
  result := FROM + AddLeadingBrackets(lResult);
end;

//=============================================================================
function TExporter.BuildMeasurement(const iBiotope: boolean): string;
begin
  if iBiotope then
    result := BIO_MEASUREMENT + '"' + FMeasurementUnitKey + '"'
  else
    result := TAX_MEASUREMENT + '"' + FMeasurementUnitKey + '"';
end;

//=============================================================================
function TExporter.BuildPref(const iBiotope: boolean): string;
begin
  if iBiotope then
    result := BIO_PREF
  else
    result := TAX_PREF;
end;

//=============================================================================
function TExporter.BuildSelect(const iBiotope: boolean): string;
begin
  result := '';
  if iBiotope then begin
    result := BIOTOPE_SELECT;
    if FMeasurementUnitKey <> '' then
      result := result + BIO_MEAS_SELECT;
  end else begin
    result := TAXON_SELECT;
    if FMeasurementUnitKey <> '' then
      result := result + TAX_MEAS_SELECT;
  end;
end;
```

```
//============================================================================
function TExporter.BuildSort(const iBiotope: boolean): string;
begin
  result := '';
  if iBiotope then
    result := BIOTOPE_SORT
  else
    result := TAXON_SORT;
end;

//============================================================================
function TExporter.BuildWhere(const iListType : TListType; const iKeys : string;
            const iBiotope : Boolean): string;
begin
  result := '';
  if iKeys <> '' then begin
    case iListType of
    ltBiotope:  result := BIOTOPE_WHERE + '(' + iKeys + ')';
    ltTaxon:    result := TAXON_WHERE + '(' + iKeys + ')';
    ltLocation: result := LOCATION_WHERE + '(' + iKeys + ')';
    ltName:
    begin
      if iBiotope then
        result := BIO_DET_WHERE + '(' + iKeys + ')'
      else
        result := TAX_DET_WHERE + '(' + iKeys + ')';
    end;
    ltSurvey:   result := SURVEY_WHERE + '(' + iKeys + ')';
    ltEvent:    result := EVENT_WHERE + '(' + iKeys + ')';
    ltSample:   result := SAMPLE_WHERE + '(' + iKeys + ')';
    end;
  end;
end;

//============================================================================
function TExporter.ExportFile(const iItemsToExport: IKeyList;
  const iDestinationFile: WideString): WordBool;
// This function actually produces the *.DIS and *.NAM files
// required by DMAP
var
  lslDISFile, lslNAMFile:TStringList;
  lIndex : integer;
  lKeys : TCommaSeparatedKeys;
  lListType : TListType;
begin
  FTypeErrors := False;
  result := false;
  try
    lslDISFile:=TStringList.Create;
    lslNAMFile:=TStringList.Create;
    try
      lKeys := BuildCommaSeparatedList(iItemsToExport);
      lListType := lKeys.ListType;
      OpenDatabase;
      lIndex := 0;
      case lListType of
      ltMixed:
        begin
        ExportOccurrences(True, ltBiotope, lKeys.Keys1, lslNAMFile, lslDISFile,
   lIndex);  //Biotope Occurrences
        ExportOccurrences(False, ltTaxon, lKeys.Keys2, lslNAMFile, lslDISFile,
   lIndex); //Taxon Occurrences
```

```
        end;
      ltLocation, ltName, ltSurvey, ltEvent, ltSample :
        begin
        ExportOccurrences(True, lListType, lKeys.Keys1, lslNAMFile, lslDISFile,
   lIndex);  //Biotope Occurrences
        ExportOccurrences(False, lListType, lKeys.Keys1, lslNAMFile, lslDISFile,
   lIndex); //Taxon Occurrences
        end;
      ltBiotope :
        begin
        ExportOccurrences(True, lListType, lKeys.Keys1, lslNAMFile, lslDISFile,
   lIndex);  //Biotope Occurrences
        end;
      ltTaxon :
        begin
        ExportOccurrences(False, lListType, lKeys.Keys1, lslNAMFile, lslDISFile,
   lIndex); //Taxon Occurrences
        end;
      end;
      lslDISFile.add('-1');
      lslDISFile.SaveToFile(RemoveExistingExt(iDestinationFile) + DIS_FILE_EXT);
      lslNAMFile.SaveToFile(RemoveExistingExt(iDestinationFile) + NAM_FILE_EXT);
      result:=true;
      if FTypeErrors then
        MessageDlg('Certain items could not be exported due to type conversion
   errors.', mtInformation, [mbOK], 0)
      else
        MessageDlg('Successfully completed DMap export.', mtInformation, [mbOK], 0);
    except
      on E:Exception do
        MessageDlg('An unknown error has occurred creating the DMap export files.',
   mtWarning, [mbOK], 0);
    end;
  finally
    lslDISFile.free;
    lslNAMFile.free;
    CloseDatabase;
  end;
end;

//============================================================================
procedure TExporter.ExportOccurrences(const iBiotope: boolean;
          const iListType : TListType; const iKeys : string;
          var ioNAMFile, ioDISFile : TStringList; var ioIndex : integer);
var
  lName, lSpatialRefString : string;
  lqryDMapExport : TJNCCQuery;
  lItemToAdd, lAddValue, lDataValue : string;
  ltfAdd : Boolean;
begin
  if iKeys <> '' then begin { check there is something to export }
    try
      lqryDMapExport := TJNCCQuery.Create(nil);
      lqryDMapExport.SessionName := FdbMain.SessionName;
      lqryDMapExport.DatabaseName := FdbMain.DatabaseName;
      with lqryDMapExport do begin
        sql.clear;
        sql.add(BuildSelect(iBiotope));
        sql.add(BuildFrom(iListType, iBiotope));
        sql.add(BuildPref(iBiotope));
        sql.add(BuildWhere(iListType, iKeys, iBiotope));
        if FMeasurementUnitKey <> '' then
          sql.add(BuildMeasurement(iBiotope));
```

```
        sql.add(BuildSort(iBiotope));
        try
          open;
          first;
          lName := '';
          while not eof do begin
            ltfAdd := True;
            lSpatialRefString:=FieldByName('SPATIAL_REF').AsString;
            lItemToAdd := RemoveSpaces(lSpatialRefString);
            if FMeasurementUnitKey <> '' then begin
              lDataValue := FieldByName('DATA').AsString;
              try
                lAddValue := SetRangeValue(StrToFloat(lDataValue));
                if lAddValue <> '' then
                  lItemToAdd := lItemToAdd + ' ' + lAddValue;
              except
                on E:EConvertError do begin
                  ltfAdd := False;
                  FTypeErrors := True;
                end else
                  raise EExporterError.Create(EST_ADD_VALUE);
              end;
            end;
            if ltfAdd then begin
              if lName <> FieldByName('NAME_FIELD').AsString then begin
                inc(ioIndex);
                lName := FieldByName('NAME_FIELD').AsString;
                ioNAMFile.add(Inttostr(ioIndex) + ' ' + lName);
                if ioIndex > 1 then ioDISFile.add('-1');
                ioDISFile.add(InttoStr(ioIndex));
              end;  //if
              ioDISFile.add(lItemToAdd);
            end;
            next;
          end;
        finally
          close;
        end;
      end;
    finally
      lqryDMapExport.Free;
    end;
  end;
end;


//==============================================================================
function TExporter.Get_DefaultFileExtension: WideString;
begin
  result := '';
end;

//==============================================================================
function TExporter.Get_Description: WideString;
begin
  result := 'COM object which provides the export functionality from Recorder 2000 to
   DMAP';
end;

//==============================================================================
function TExporter.Get_ImageFileName: WideString;
begin
  result := '';
```

```
end;

//=============================================================================
function TExporter.Get_LastExportError: WideString;
begin
  result := '';
end;

//=============================================================================
function TExporter.Get_MaxValue: Double;
begin
  result := FMaxValue;
end;

//=============================================================================
function TExporter.Get_MeasurementUnitKey: WideString;
begin
  result := FMeasurementUnitKey;
end;

//=============================================================================
function TExporter.Get_MinValue: Double;
begin
  result := FMinValue;
end;

//=============================================================================
function TExporter.Get_Name: WideString;
begin
  result := 'DMAP export';
end;

//=============================================================================
function TExporter.Get_NumberOfRanges: Integer;
begin
  result := FNumberOfRanges;
end;

//=============================================================================
procedure TExporter.Install;
begin
  showmessage('The DMap Exporter has been successfully installed');
end;

//=============================================================================
procedure TExporter.OpenDatabase;
var
  lReg:TRegistry;
  lDatabasePath, lDatabasePassword:string;
begin
  lReg := TRegistry.Create;
  with lReg do
    try
      // Get global Settings
      if OpenKey('Software\JNCC\Recorder\Settings', False) then begin
        if ValueExists('Database Path') then
          lDatabasePath:=ReadString('Database Path')
        else
          raise EExporterError.Create(EST_DBPATH_MISSING);
        if ValueExists('Database Password') then
          lDatabasePassword:=ReadString('Database Password')
        else
          raise EExporterError.Create(EST_PASSWORD_MISSING);
```

```
        CloseKey;
      end;  // if lReg.OpenKey...
    finally
      Free;
    end;
  FssMain := TtaSession.Create(nil);
  FssMain.SessionName := JNCC_SESSION;
  FssMain.Active := True;
  FdbMain := TtaDatabase.Create(nil);
  FdbMain.SessionName := JNCC_SESSION;
  FdbMain.DatabaseName := JNCC_DATABASE;
  FdbMain.AliasName := lDatabasePath;
  if lDatabasePassword <> '' then
     FdbMain.Params.Add('PASSWORD=' + lDatabasePassword);
  try
     FdbMain.Open;
  except
   on E:ETDBEngineError do
       if E.Errors[0].ErrorCode = 10015 then begin
          MessageDlg(EST_WRONG_PASSWORD, mtError, [mbOK], 0);
        Raise;
       end else
         Raise EExporterError.Create(EST_OPEN_FAIL);
    else
      Raise EExporterError.Create(EST_OPEN_FAIL);
  end;
end;  // OpenDatabase

//==============================================================================
function TExporter.RemoveExistingExt(const iFile: string): string;
var
  lPos : integer;
  lResult, lRemainder : string;
begin
  lResult := '';
  lRemainder := iFile;
  lPos := Pos('.', lRemainder);
  if lPos = 0 then
    Result := lRemainder
  else begin
    while lPos > 0 do begin
      lResult := lResult + Copy(lRemainder, 1, lPos);
      lRemainder := Copy(lRemainder, lPos + 1, length(lRemainder) - lPos);
      lPos := Pos('.', lRemainder);
    end;

    //Check if not a DMap file
    if (CompareText(lRemainder, Copy(DIS_FILE_EXT, 2, length(DIS_FILE_EXT) - 1)) <> 0)
   and
       (CompareText(lRemainder, Copy(NAM_FILE_EXT, 2, length(NAM_FILE_EXT) - 1)) <> 0)
   then
      lResult := lResult + lRemainder;

    //Check if ends in .
    if lResult[length(lResult)] = '.' then
      lResult := Copy(lResult, 1, length(lResult) - 1);

    Result := lResult;
  end;
end;

//==============================================================================
function TExporter.RemoveSpaces(const iString: string): string;
```

Formatted: English (U.K.)

```
var
  lSpacePos : integer;
  lResult : string;
begin
  lResult := iString;
  lSpacePos := Pos(' ', lResult);
  while lSpacePos > 0 do begin
    lResult := Copy(lResult, 1, lSpacePos - 1) + Copy(lResult, lSpacePos + 1,
   length(lResult) - lSpacePos);
    lSpacePos := Pos(' ', lResult);
  end;
  Result := lResult;
end;

//============================================================================
function TExporter.SetRangeValue(const iValue: double): string;
var
  lIndex : integer;
  lTotalRange, lSingleRange, lCurrentRangeStart, lCurrentRangeEnd : extended;
begin
  Result := '';
  if FNumberOfRanges > 2 then begin
    lTotalRange := FMaxValue - FMinValue;
    lSingleRange := lTotalRange / (FNumberOfRanges - 2);

    if iValue <= FMinValue then begin
      Result := IntToStr(FNumberOfRanges);
      Exit;
    end;

    if iValue > FMaxValue then begin
      Result := '1';
      exit;
    end;

    //Check intermediate ranges
    for lIndex := 0 to FNumberOfRanges - 3 do begin
      lCurrentRangeStart := (lIndex * lSingleRange) + FMinValue;
      lCurrentRangeEnd := lCurrentRangeStart + lSingleRange;
      if (iValue > lCurrentRangeStart) and (iValue <= lCurrentRangeEnd) then begin
        Result := IntToStr(FNumberOfRanges -1 - lIndex);
        Exit;
      end;
    end;
  end else begin
    if iValue <= FMinValue then
      Result := '2'
    else
      Result := '1';
  end;
end;

//============================================================================
procedure TExporter.SetRecorder2000Interface(
  const iRec2000Intf: IRecorder2000);
begin
  FRecorder2000 := iRec2000Intf;
end;

//============================================================================
procedure TExporter.Set_MaxValue(Value: Double);
begin
  FMaxValue := Value;
```

Formatted: English (U.K.)

```
end;

//=============================================================================
procedure TExporter.Set_MeasurementUnitKey(const Value: WideString);
begin
  FMeasurementUnitKey := Value;
end;

//=============================================================================

procedure TExporter.Set_MinValue(Value: Double);
begin
  FMinValue := Value;
end;

//=============================================================================
procedure TExporter.Set_NumberOfRanges(Value: Integer);
begin
  FNumberOfRanges := Value;
end;

//=============================================================================
function TExporter.TrimFirstWord(const iString: string): string;
var
  lPos : integer;
begin
  lpos := Pos(' ', iString);
  result := Copy(iString, lPos+1, length(iString) - lPos);
end;

//=============================================================================
procedure TExporter.CloseDatabase;
begin
  FdbMain.Close;
  FdbMain.Free;
  FssMain.Active := False;
  FssMain.Free;
end;

//=============================================================================
initialization
  TComObjectFactory.Create(ComServer, TExporter, Class_Exporter,
    'Exporter', 'COM object which provides the export functionality from Recorder 2000
   to DMAP', ciMultiInstance, tmApartment);
end.
```

SUPPORTING RECORDER 2000 EXPOSED PROPERTIES THROUGH COM OBJECTS

Recorder 2000 exposes certain properties through the TAutoApplicationSettings object which implements the IRecorder2000, IRucksack and ICurrentSettings interfaces. These properties allow add-in modules to interact with Recorder 2000 so that they can use certain items such as the currently selected rucksack to provide them with key, global information. Add-in modules can automate this part of the global Recorder 2000 settings and request information through the IRucksack and ICurrentSettings interfaces.

The following example shows how the taxon list in the currently selected rucksack in Recorder 2000 is available to all COM objects supporting the IRucksack interface.

1. Create a new Active form which supports the IRecorderAddIn and INewAction interfaces. See listing 3.1 below for an example of an Active Form which automates Recorder2000.AutoApplicationSettings and uses this to obtain the contents of the taxon list in the current rucksack.

listing 3.1

```
unit SupportRucksack;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, DMAPServer_TLB, Recorder2000_TLB, StdCtrls, Buttons;

type
  TSupportRucksack = class(TActiveForm, ISupportRucksack, IRecorderAddIn, INewAction)
    lbTaxonList: TListBox;
    lblTaxonList: TLabel;
    bbPopList: TBitBtn;
    procedure bbPopListClick(Sender: TObject);
  private
    { Private declarations }
    FEvents: ISupportRucksackEvents;
    FRecorderRucksack: IRucksack;

    procedure ActivateEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure CreateEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure DeactivateEvent(Sender: TObject);
    procedure DestroyEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
    procedure PaintEvent(Sender: TObject);
  protected
    { Protected declarations }
```

```
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    function Get_Active: WordBool; safecall;
    function Get_AutoScroll: WordBool; safecall;
    function Get_AutoSize: WordBool; safecall;
    function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
    function Get_BiDiMode: TxBiDiMode; safecall;
    function Get_Caption: WideString; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_Cursor: Smallint; safecall;
    function Get_DoubleBuffered: WordBool; safecall;
    function Get_DropTarget: WordBool; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_Font: IFontDisp; safecall;
    function Get_HelpFile: WideString; safecall;
    function Get_KeyPreview: WordBool; safecall;
    function Get_PixelsPerInch: Integer; safecall;
    function Get_PrintScale: TxPrintScale; safecall;
    function Get_Scaled: WordBool; safecall;
    function Get_Visible: WordBool; safecall;
    procedure _Set_Font(const Value: IFontDisp); safecall;
    procedure Set_AutoScroll(Value: WordBool); safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
    procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
    procedure Set_Caption(const Value: WideString); safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DropTarget(Value: WordBool); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(var Value: IFontDisp); safecall;
    procedure Set_HelpFile(const Value: WideString); safecall;
    procedure Set_KeyPreview(Value: WordBool); safecall;
    procedure Set_PixelsPerInch(Value: Integer); safecall;
    procedure Set_PrintScale(Value: TxPrintScale); safecall;
    procedure Set_Scaled(Value: WordBool); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    { IRecorderAddIn }
    function Get_Name: WideString; safecall;
    function Get_Description: WideString; safecall;
    function Get_ImageFileName: WideString; safecall;
    procedure Install; safecall;
    { INewAction }
    function Get_ActionCaption: WideString; safecall;
    function Get_Hint: WideString; safecall;
    function Get_DimmedImageFilename: WideString; safecall;
    function Get_DisabledImageFileName: WideString; safecall;
    function Get_ParentMenu: WideString; safecall;
    function Get_CanAddToToolbar: WordBool; safecall;
  public
    { Public declarations }
    procedure Initialize; override;

  end;

implementation

uses ComObj, ComServ, DataClasses;

{$R *.DFM}

{ TSupportRucksack }
```

```
procedure TSupportRucksack.DefinePropertyPages(DefinePropertyPage:
   TDefinePropertyPage);
begin
  { Define property pages here.  Property pages are defined by calling
    DefinePropertyPage with the class id of the page.  For example,
      DefinePropertyPage(Class_SupportRucksackPage); }
end;

//============================================================================

procedure TSupportRucksack.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as ISupportRucksackEvents;
end;

//============================================================================

procedure TSupportRucksack.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
end;

//============================================================================

function TSupportRucksack.Get_Active: WordBool;
begin
  Result := Active;
end;

//============================================================================

function TSupportRucksack.Get_AutoScroll: WordBool;
begin
  Result := AutoScroll;
end;

//============================================================================

function TSupportRucksack.Get_AutoSize: WordBool;
begin
  Result := AutoSize;
end;

//============================================================================

function TSupportRucksack.Get_AxBorderStyle: TxActiveFormBorderStyle;
begin
  Result := Ord(AxBorderStyle);
end;

//============================================================================

function TSupportRucksack.Get_BiDiMode: TxBiDiMode;
begin
```

```
  Result := Ord(BiDiMode);
end;

//=============================================================================

function TSupportRucksack.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

//=============================================================================

function TSupportRucksack.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(Color);
end;

//=============================================================================

function TSupportRucksack.Get_Cursor: Smallint;
begin
  Result := Smallint(Cursor);
end;

//=============================================================================

function TSupportRucksack.Get_DoubleBuffered: WordBool;
begin
  Result := DoubleBuffered;
end;

//=============================================================================

function TSupportRucksack.Get_DropTarget: WordBool;
begin
  Result := DropTarget;
end;

//=============================================================================

function TSupportRucksack.Get_Enabled: WordBool;
begin
  Result := Enabled;
end;

//=============================================================================

function TSupportRucksack.Get_Font: IFontDisp;
begin
  GetOleFont(Font, Result);
end;

//=============================================================================

function TSupportRucksack.Get_HelpFile: WideString;
begin
  Result := WideString(HelpFile);
end;

//=============================================================================

function TSupportRucksack.Get_KeyPreview: WordBool;
begin
```

```
  Result := KeyPreview;
end;

//============================================================================

function TSupportRucksack.Get_PixelsPerInch: Integer;
begin
  Result := PixelsPerInch;
end;

//============================================================================

function TSupportRucksack.Get_PrintScale: TxPrintScale;
begin
  Result := Ord(PrintScale);
end;

//============================================================================

function TSupportRucksack.Get_Scaled: WordBool;
begin
  Result := Scaled;
end;

//============================================================================

function TSupportRucksack.Get_Visible: WordBool;
begin
  Result := Visible;
end;

//============================================================================

procedure TSupportRucksack._Set_Font(const Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;

//============================================================================

procedure TSupportRucksack.Set_AutoScroll(Value: WordBool);
begin
  AutoScroll := Value;
end;

//============================================================================

procedure TSupportRucksack.Set_AutoSize(Value: WordBool);
begin
  AutoSize := Value;
end;

//============================================================================

procedure TSupportRucksack.Set_AxBorderStyle(
  Value: TxActiveFormBorderStyle);
begin
  AxBorderStyle := TActiveFormBorderStyle(Value);
end;

//============================================================================

procedure TSupportRucksack.Set_BiDiMode(Value: TxBiDiMode);
```

```
begin
  BiDiMode := TBiDiMode(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_Caption(const Value: WideString);
begin
  Caption := TCaption(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_Color(Value: OLE_COLOR);
begin
  Color := TColor(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_Cursor(Value: Smallint);
begin
  Cursor := TCursor(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_DoubleBuffered(Value: WordBool);
begin
  DoubleBuffered := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_DropTarget(Value: WordBool);
begin
  DropTarget := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_Enabled(Value: WordBool);
begin
  Enabled := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_Font(var Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_HelpFile(const Value: WideString);
begin
  HelpFile := String(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_KeyPreview(Value: WordBool);
```

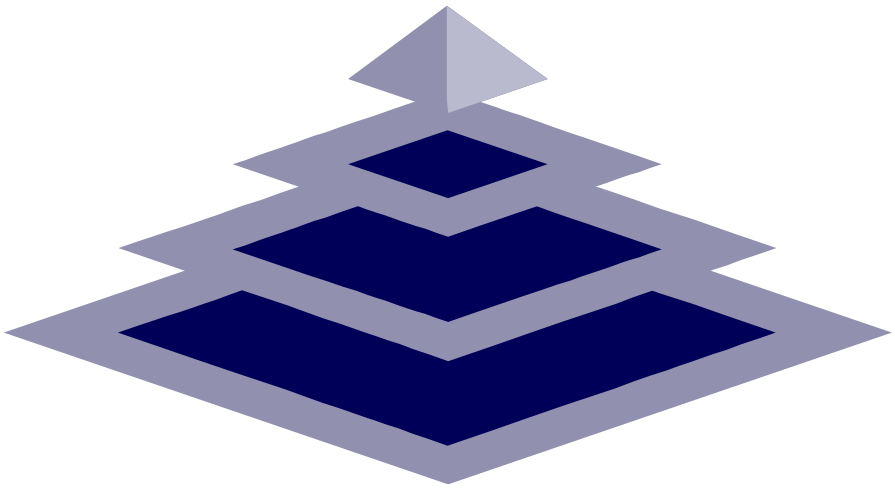
```
begin
  KeyPreview := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_PixelsPerInch(Value: Integer);
begin
  PixelsPerInch := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_PrintScale(Value: TxPrintScale);
begin
  PrintScale := TPrintScale(Value);
end;

//=============================================================================

procedure TSupportRucksack.Set_Scaled(Value: WordBool);
begin
  Scaled := Value;
end;

//=============================================================================

procedure TSupportRucksack.Set_Visible(Value: WordBool);
begin
  Visible := Value;
end;

//=============================================================================

procedure TSupportRucksack.ActivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnActivate;
end;

//=============================================================================

procedure TSupportRucksack.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

//=============================================================================

procedure TSupportRucksack.CreateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnCreate;
end;

//=============================================================================

procedure TSupportRucksack.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

//=============================================================================

procedure TSupportRucksack.DeactivateEvent(Sender: TObject);
```

```
begin
  if FEvents <> nil then FEvents.OnDeactivate;
end;

//==============================================================================

procedure TSupportRucksack.DestroyEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDestroy;
end;

//==============================================================================

procedure TSupportRucksack.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

//==============================================================================

procedure TSupportRucksack.PaintEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnPaint;
end;

//==============================================================================

procedure TSupportRucksack.bbPopListClick(Sender: TObject);
var
  lCount, lTotal : integer;
  lCurrentListItem : IKeyItem;
  lListKey : TKeyString;
  lTaxonList : IKeyList;
  lObj : IDispatch;
begin
  lObj := CreateOLEObject('Recorder2000.AutoApplicationSettings');
  FRecorderRucksack:=lObj as IRucksack;
  lTaxonList := FRecorderRucksack.Get_TaxonList;
  lTotal := lTaxonList.ItemCount;
  for lCount := 0 to lTotal - 1 do begin
    lCurrentListItem := lTaxonList.GetKeyItem(lCount);
    lListKey := lCurrentListItem.KeyField1;
    lbTaxonList.Items.Add(lListKey);
  end; // for
end;

//==============================================================================

function TSupportRucksack.Get_Description: WideString;
begin
  result := 'Test object for reading the contents of the rucksack';
end;

//==============================================================================

function TSupportRucksack.Get_ImageFileName: WideString;
begin
  result := '';
end;
```

```
//=============================================================================

function TSupportRucksack.Get_Name: WideString;
begin
  result := 'Support Rucksack';
end;

//=============================================================================

function TSupportRucksack.Get_ActionCaption: WideString;
begin
  result := 'Support';
end;

//=============================================================================

function TSupportRucksack.Get_CanAddToToolbar: WordBool;
begin
  result := false;
end;

//=============================================================================

function TSupportRucksack.Get_DimmedImageFilename: WideString;
begin
  result := '';
end;

//=============================================================================

function TSupportRucksack.Get_DisabledImageFileName: WideString;
begin
  result := '';
end;

//=============================================================================

function TSupportRucksack.Get_Hint: WideString;
begin
  result := 'Test object for rucksack';
end;

//=============================================================================

function TSupportRucksack.Get_ParentMenu: WideString;
begin
  result := 'File';
end;

//=============================================================================

procedure TSupportRucksack.Install;
begin
  showmessage('Install');
end;

//=============================================================================

initialization
  TActiveFormFactory.Create(
    ComServer,
    TActiveFormControl,
```

```
    TSupportRucksack,
    Class_SupportRucksack,
    1,
    '',
    OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
    tmApartment);
end.
```

2. Recorder 2000 contains an ole container form, frmOLEContainer, detailed in the TSD, which uses TOLEProxy, again defined in the TSD, to provide an MDI child form in which to house Active Forms / ActiveXControls. This is due to the fact that Active forms themselves cannot be MDI. This form contains a procedure that takes all properties of ICurrentSettings as parameters to allow user's to pass the current settings from Recorder 2000 to the Active Form.



Corbiere House, New Quay Road, Poole, Dorset, BH15 4AF, U.K.

Telephone: 01202 777707 Facsimile: 01202 774016

E-Mail: sales@dorsetsoftware.com