1 Table of Contents

2 Welcome

This documentation describes the API for Mindjet 11 for Windows, and highlights changes and additions in the API between MindManager 2012 and Mindjet 11.

This is build 642 of the API documentation. The Object Model documentation reflects the Object Model version 11.2.1 as published at Mindjet 11 v11.2.185.

This documentation is written for developers who are using Mindjet 11 for Windows as a platform for an information management solution, or to develop extensions to Mindjet 11 for other users inside or outside their organization. Some knowledge of software development environments, programming techniques and software design are assumed and are outside the scope of this documentation. It is designed to be used alongside the documentation for your development environment and programming language, unless you are working exclusively with Mindjet 11's built-in macro language, WinWrap Basic.

The documentation consists of:

- Conceptual information, describing at a high level principles such as where and how to use add-ins, iterating over the contents of a map, or storing custom data in maps, and
- Reference information for the object model, describing the objects, methods, properties and events that make up the COM API, with
- Special notes (advisories) that affect the practical use of the API or which might need additional consideration

Throughout this document, references to "Mindjet 11" explicitly refer to **Mindjet 11 for Windows** only. The API description does not apply to other Mindjet products for other platforms, such as web, Mac or mobile applications. Where available, the APIs for Mindjet products on other platforms are documented separately.

This documentation aims to provide a solid foundation for developers, without providing detailed examples for every single entity in the Mindjet 11 object model. As there are over 2000 objects, methods and properties in the object model, only the most frequently used items (around 15%) have extended documentation and sample code, although all are referenced here. Once the basic concepts and patterns are grasped, most developers will find it straightforward to extend this knowledge to less frequently used but similar entities without requiring detailed documentation.

This format of the documentation (PDF booklet) contains the conceptual overviews and tutorials, but does not contain the detailed documentation of the object model entities. These are available together with sample code in the CHM and online formats of this documentation.

Updating from previous versions

If you are updating from an older version of MindManager, refer to the **Appendix ('Changes by platform version' in the on-line documentation)** on changes by platform version for specific details. The Appendices and Index list version-specific changes.

3 Why Mindjet 11?

3.1 Mindjet 11 Capabilities

Mindjet 11 is the market leading "mind mapping" and information visualization software tool with a large user base and extensive corporate deployment. It is also one of only a small number which are capable of extension and automation via a published API. This means that in addition to the built-in feature set delivered by Mindjet, Mindjet 11 for Windows is a platform for visualization of information of all kinds. Using the API, Mindjet 11 can drill into databases, perform proprietary data transformations, or create tools for analyzing and presenting information.

For organizations working with intellectual property and knowledge, Mindjet 11 provides a low cost of entry to customized information analysis and display, bringing the benefits of tree diagrams to help with understanding information and taking action on it. Mindjet 11 also offers a unique combination of automatically derived and structured content with user-defined free form content in the same view, creating a flexible and productive environment for manipulating information.

3.2 Application Scenarios

3.2.1 Exchanging Data with Import and Export

The Mindjet 11 API can be used to implement new imports and exports, converting data in other formats to and from Mindjet 11 maps. For example, a command could be added to Mindjet 11 to export the current document as a database file that can be opened with another program. The API provides the capability to add a new command to a menu or ribbon, access the contents of a map, and structure & write a new file. Usually, the challenge with imports and exports is defining the transformation between hierarchical data and the format in the target application.

There are two ways to perform imports and exports; programmatic, using code to interpret and structure the target data, or by applying an XSLT transformation to the XML of the map. Mindjet 11 supports both methods for both import and export.

3.2.2 Integration with Other Systems

The API can be used to connect Mindjet 11 in real time to other systems, sending and receiving data automatically or on demand. For example, you could keep a special element in a map synchronized with corresponding data in another system over a web service. Changes in the remote system would be reflected in the map, and vice versa.

The Mindjet 11 API has a rich set of events, including a background "idle" event, that can be used to detect local changes or poll remote information. The user interface is not in exclusive control of maps, which can be automatically updated in response to other triggers. The Mindjet 11 platform can become an interface to another system, interpreting data and events in the map.

3.2.3 Extending Mindjet 11's Functionality

The third way that the API can be used is to add new features and functions to the user interface to perform a local function on the current map, or generate new maps. For example, you could add a command to Mindjet 11 to apply colors to action topics by the resource name assigned.

The API allows commands to be added to menus and ribbon tabs, and new Task Panes to be created for editing and displaying information. You cannot use the API to remove or modify an existing Mindjet 11 feature or function, but new alternatives can be added, and in many cases an existing function (such as closing or saving a map) can be extended by handling the associated event.

3.3 Strategies And Skills

Solutions built using the Mindjet 11 for Windows API fall into two main areas:

- In-house applications, where you develop an extension or special purpose function for use by your organization. For example, you may develop a tool that supports a proprietary business process. This could be delivered by an external consultant, or you may have in-house IT resources who can take this on. There is usually a tight coupling between the designers and the users, and frequent changes in the specification are common. The number of environments and configurations supported can be controlled, and some "rough edges" can be tolerated if the audience is a trusted one. Licensing and security are not usually an issue.
- Published applications, where you develop an extension and publish it for use outside your organization. This is a very different environment as many of the factors are outside your control, but with a platform user base approaching 2 million users, there are many opportunities for vertical market applications of Mindjet 11 for Windows. Security, trial & licensing, localization, support and packaging are all factors that need to be considered when delivering a product as opposed to a project. Entry level costs and learning curves can be high for users who are not already Mindjet 11 users, so consulting organizations are often better placed to deploy into green field sites than software developers or resellers. An existing Mindjet 11 user base is a better starting point.

The skill sets needed to deliver a solution that uses the Mindjet 11 for Windows API depend on the technology used:

- For all types of solution, in-depth familiarity with Mindjet 11 functionality, mapping techniques and the Mindjet 11 user interface are essential.
- For all types of solution, a working knowledge of the Mindjet 11 Object Model is essential.
- For internal solutions delivered using Mindjet 11 Macros, only a working knowledge of WinWrap Basic (which is VBA compatible) is needed, and no other tools are necessary.
- For solutions that use Mindjet 11 Add-ins, some knowledge of the Microsoft Office Extensibility API is required, together with a suitable development environment such as Microsoft Visual Studio to generate DLLs.
- Knowledge of XML and XSLT optionally applies to both, as XML transformations can be deployed in either environment.
- To deliver a product for sale, knowledge of localization, licensing and packaging technologies will be required. These are outside the scope of this documentation.

4 Programming Concepts

4.1 Glossary of terms

The entities in the API are mostly named after the related Mindjet 11 feature, but the external names can change over time while the named entities in the API are fixed. There are also some API entities and concepts that are not externally visualized.

The descriptions below cover the main vocabulary in the Mindjet 11 for Windows API, and link the objects in the API to their external features.

Add-in: a registered DLL that exposes the Microsoft Office IDTExtensibility2 interface, which allows Mindjet 11 to connect to it and disconnect from it. Add-ins can then register themselves as an event listener, and receive events that relate to changes in the map or actions in the UI (such as clicking a button that is owned by the Add-in). Add-ins usually add new commands, task panes and other functionality to Mindjet 11. They exit when the MindManager.exe process quits, and do not have an independent process or UI outside of Mindjet 11.

Application: the running instance of Mindjet 11, which owns the API. Only one instance of Mindjet 11 can run at a time, and unless opened invisibly, is usually associated with a visible UI.

Bookmark: a deprecated feature from earlier MindManager versions, now superseded by **Topic Labels**.

Boundary: a filled shape drawn around a topic and its subtopics in a map. Boundaries can be nested, but not overlapped. Boundaries can also have callout topics and relationships attached.

Branch: a commonly used term, but the correct name in MindManager / Mindjet is **Topic**.

BusinessTopic: a topic that carries additional user-editable data. The characteristics of a BusinessTopic are a logo icon, an associated context menu, and a display/edit area for data that can be expanded, resized or collapsed. A BusinessTopic can be one of three basic types: Custom Properties, Tables, or Custom.

Callback: a handler routine within Add-in code that is invoked by Mindjet 11 when an event for which the Add-in has registered is raised. Callbacks are used for editing events, running transactions, events in the Topic Finder, and updating the ribbon.

Callout Topic: a **Topic** that is attached to a parent topic, but is not part of the regular subtopics collection belonging to that parent. Callout topics can be independently viewed or hidden in the View > Show/Hide menu in the UI, although this feature is not exposed in the API. Callout topics can also be attached to **Boundaries** and **Relationships**.

Central Topic: the topic that is at the centre of the map. Every map has one and only one central topic. If the map is being viewed in "Show Branch Alone" mode, the visible central topic may not be the original central topic in the map. A separate property for the visible central topic is available.

Child (Topic): see **Subtopic**.

Command: an object that can be executed to perform a function. A command is distinct from a button, which is visualized. Buttons, menu items and control strip icons use commands to execute code.

Control Strip: a group of special icons that are not part of the map marker set, but which can be displayed on a topic and can be clicked by the user to perform custom functions.

Custom Attributes: storage of named text data fields associated with various elements of a map. Topics, map markers, map marker sets and documents can all have attributes. An attribute has a name and a text value, and belongs to an attribute namespace, which has a unique URI. Attributes are persistent and are stored with the document, so can be used to retain custom data between editing sessions. Attributes are stored as XML, so care should be taken with white space as this may not be accurately preserved.

Custom BusinessTopic: a type of BusinessTopic where the Add-in or external code that registers and owns the BusinessType is responsible for rendering the visualization in the topic, by embedding a form in the topic. This allows almost any kind of custom controls to be added to a BusinessTopic.

Custom Properties (BusinessTopic): a set of data values associated with a topic, that can be edited by the user in the Mindjet 11 UI. The values are shown in a grid underneath the topic. The grid can be expanded or collapsed with an arrow icon on the topic itself. Custom Properties are a specific type of Business Topic. In Mindjet 11, Custom Properties are called "Topic Properties" in the UI, although they are still called Custom Properties inside the Object Model.

Document: a map or related document displayed in a single MDI child window. Maps are the most commonly used documents, but a document may also refer to a map part, a template, a map marker set or a map style. Only maps and map parts can be opened in a document window. There is always one active document, which is the document window that has focus.

Document Bar: a form which is docked on one edge of a document window, and contains controls for document-specific functions. Document bars are similar to a task pane, but are unique to a document, and are always visible if present. Document Bars can be docked above, below, to the left or to the right of the working map. Because they are document-specific, they do not need to refresh when another document is activated. Document bars are not persistent (i.e. not stored in the document itself) and can be turned on and off by add-ins. The Add-in that creates a document bar is responsible for editing it and handling its controls and for destroying it when the document closes.

Document Object: an artifact in a map, i.e. a topic, a relationship or a boundary. All maps are made up from these three components.

Edition: some versions of MindManager were available in different editions, such as Professional or Standard. Although enumerations still exist for Professional and Standard editions, they do not reflect any difference in functionality in later versions of MindManager / Mindjet.

EMF: The file extension for Extended Meta Files, the vector graphics file type used for topic shapes in MindManager / Mindjet.

Event: Add-ins can register as event handlers and get notified of events such as documents opening and closing, topics being selected, modified, added or deleted, or system events such as the "idle" tick or ribbon button refreshing being required. Many of the events related to changes in maps have "before" and "after" instances, so that an Add-in can detect whether something in which it has an interest actually changes. For example, when a topic is edited by the user, Mindjet 11 fires a "before modification" and "after modification" event pair in succession. Note though that these pairs are not guaranteed to be non-interleaved, so "before" events should be cached and matched to the "after" event using Topic Guids. Some events are ad-hoc and occur without warning, such as an edit being made to the map, or a command button being clicked. Other events occur regularly, such as the updating of command buttons or the Idle event. Others occur only in response to actions initiated through the API, such as launching a Transaction object, or launching a TopicFinder object.

External Topic: a topic that does not load its subtopics until it is expanded by the user. When the user clicks the "+" button at the end of the topic line to expand subtopics, MindManager invokes a callback routine to expand them. This is convenient for dynamic results that might take some extra time to load, as it allows the initial map to load quickly without exploring and capturing all possible lines of drill-down in advance. An External Topic handler is allocated and associated with a callback routine, then topics are added to that handler.

Floating Topic: a topic that is not connected to the main tree, but which sits at a fixed location in the background. Floating topics can be displayed or hidden, configured in the View > Show/Hide menu in the UI, but this option is not available through the API.

Frame: a frame is a rarely-used device which draws a rectangle around a topic in the map, and can be used to show a temporary status. Frames are not persistent and are not stored in the document data.

Guid: Document objects (e.g. Topics) have a unique Guid that is stored in the object and is persistent between sessions. If you need to identify a topic in a map, even when it changes location, consider using the Guid. The API offers a method for finding any topic from its Guid. Guids are also used to make hyperlinks to topics either within the document or from the outside. Note that Guids will change if the topic is copied or if the XML of a topic is overwritten.

Icon: see Map Marker.

Ink and Sketches: if Mindjet 11 is running on a Windows Tablet PC, ink and sketches attached to a topic can be accessed through the API, together with provisional recognition text. For maximum usability, the "text" of a topic should be taken as the plain text if it is different to the default new-topic setting, but if the topic text remains at the default setting then any provisional recognition text from Ink should be used instead.

Keyboard Shortcut: Most of the commands in the Mindjet 11 ribbon have an associated keyboard shortcut sequence, comprised of the Alt key and a single character to select the tab, then one or two keys for the command within that tab. There is no central registry for keyboard shortcuts, and you may find that if you adopt certain characters for your Add-in commands, then they will either clash in other localizations, with other Add-ins, or may be overridden by a future version of Mindjet 11.

Macro: Mindjet 11 includes a VBA-compatible programming language (WinWrap Basic) which is suitable for scripting relatively simple functions. Mindjet 11 ships with a built-in editor & debugging tool for macros. Macros have no scope when they are not running, so are unsuitable for session-based customization such as task panes, ribbon buttons or ad-hoc event handling.

Map Marker: an icon, text/tag marker, font color, fill color, task priority, task complete value or task resource. Map Markers are organized into Map Marker Groups, which have properties such as mutual exclusion. Markers are globally unique, i.e. cannot be used more than once in different groups. When a marker is deployed on a topic, only the marker information is stored, not the group. This allows markers to be moved between groups and copied between maps without disrupting either the map or the map marker groups.

Marker Group: A collection of **Map Markers**.

MDI window: Mindjet 11 handles multiple documents within the same application by having multiple document windows, rather than a single document window and multiple applications. MDI windows can either be Mindjet 11 documents (maps, map parts or styles), or can be custom MDI windows where the content is provided by an Add-in and is not managed by Mindjet 11. MDI child windows can be floating, tiled or maximized with tabs. The tabs collection itself is not published in the API.

Mindjet Catalyst: the online file and folder storage system supported up to and including MindManager 9. From MindManager 2012 onwards this is replaced by **Mindjet Files**.

Mindjet Files: The cloud-based file storage & management feature within the Mindjet system, replacing the former Mindjet Catalyst feature. Mindjet Files can be accessed from the Windows client through the **MjService API.**

Mindjet Tasks: The cloud-based task storage and management feature within the Mindjet system. Currently there is no direct API for Mindjet Tasks within the Windows client API.

MjService API: A COM API to Mindjet Files. The MjService API is integrated into the Windows client API and allows content to be created, uploaded, checked in and checked out, and management of folders. It reflects the desktop client's version of the status of the Mindjet Files server, which must be refreshed to ensure that it stays synchronized and contains current information. The MjService API also includes new properties in the Document object that indicate the status of documents that originated from Mindjet Files.

MMAP: the file extension for MindManager / Mindjet map documents. The file is actually a zip archive containing a number of further files, including the XML sources.

MMAS: the file extension for MindManager / Mindjet map style documents, containing a map style that can be applied to an existing map.

MMBAS: the file extension for MindManager / Mindjet macros, which can be edited and executed with the built-in macro editor.

MMMP: the file extension for MindManager / Mindjet Map Parts, containing a single map part that can be embedded in another map.

MMMS: the file extension for MindManager / Mindjet Map Marker Set documents, containing a map marker set that can be applied to an existing map.

MMScript: the former name for **Macros.**

Notes (Topic Notes): a self-contained document stored as XHTML inside a Topic, and displayed & edited in the UI. Mindjet 11 offers limited programmatic access to Notes; for example, tables and font colors cannot be accessed without parsing the XHTML. As well as XHTML, plain text and RTF versions of the topic notes can be written and read. Conversion of Notes to and from XHTML is slow, and processing of large maps by accessing RTF or plain text should be avoided. The user's selection in the Notes window cannot be accessed or controlled through the API.

Object Model: the set of classes, methods and properties that are exposed through the COM Automation interface, and are published in the MindManager type library (MindManager.tlb). In Mindjet 11 for Windows, the "marketing" name is "Mindjet 11" whereas the internal object model / COM server name is "MindManager 11". In this documentation, references to the object model or COM server use "MindManager", which is how they will appear in object browsers, library references, registry keys and so on. References to the user interface use "Mindjet 11", which is what the end-user sees.

Options: the user settings for personalizing Mindjet 11 at the application level, edited through the Mindjet 11 Options dialogue. Many (but not all) of these options can be accessed through the API in the ApplicationOptions class. Options that cannot be accessed here can often be read directly from the registry, although any changes made by updating the registry will not take effect until the next restart. Add-ins can add private application-level options by registering their own Options Pane to Mindjet 11's Options dialogue.

Parent (Topic): the next topic towards the centre of the map. All topics except the central topic and floating topics have a parent topic. The parent of a callout topic is taken to be the topic to which it is attached.

Private data: Add-ins and macros can embed private persistent data at the Document level (Documents, Map Marker Groups and Map Markers) and in Document Objects (Topics, Relationships and Boundaries), using Custom Attributes. Private user-editable persistent data can also be stored at the Topic level using Custom Properties. This data will be preserved if maps transit through non-Windows platforms where these features are not exposed. However, there is no application-level storage of private data, i.e. there are no Custom Attributes for the Application Options. Add-ins and macros must use private registry values or disk files to store application-level configuration information.

Relationship: a line in the map between topics or boundaries. Relationships are owned by the document, and not by the topic or boundaries that they connect.

Ribbon: the common term for the Microsoft Fluent® User Interface used in Microsoft Office applications from Office 2007 onwards. The ribbon consists of a number of tabs, each tab being divided up into Groups containing buttons or other controls. The state of the controls in the currently displayed ribbon is maintained by calling update events for all controls that are currently visible, so that an Add-in can set controls to be enabled or disabled according to the current document and selection. Add-ins can add extra command groups to the existing Mindjet 11 ribbon tabs, but cannot directly access any commands or controls in tabs that they do not own, so it is not possible to hijack an existing feature and repurpose it.

Sax Basic: the macro language used by MindManager X5 to MindManager 2012 inclusive. See also **WinWrap Basic**.

Selection: the current selection of document objects in each document can vary from nothing selected up to a mixture of different object types, and any code that reads the user selection must cope with a range of conditions. Mindjet 11 fires an event each time the selection changes. The user selection can also be modified through the API.

Sibling (Topic): a topic that belongs to the same parent of the current topic.

Subtopic: a **Topic** connected to the current one, further away from the central topic. "Leaf" topics at the edges of the tree do not have subtopics.

Table (BusinessTopic): A BusinessType that displays a simple spreadsheet table. The spreadsheet data is stored in the topic and can be edited by the user. The contents of the spreadsheet can also be accessed programmatically through the API, allowing close data integration. (Tables in topic notes are entirely separate and there is no API support for tables in Notes.)

Tag: see **Map Marker**.

Task: a data entity attached to a Topic, which contains a number of task-related attributes such as task priority, task due date and so on. Tasks only exist within the context of a Topic, and are edited in the Task Info pane or via the API. There is no collection of Tasks in a document, so to find all tasks, all Topics must be iterated. Task information can be independently displayed or hidden in the map, configured in the View > Show/Hide menu in the UI. Note that from MindManager 9 onwards, MindManager automatically applies constraints to task information to ensure that it can be correctly displayed in Gantt charts. This means that the built-in topic task information is not always suitable for general purpose applications.

Task Pane: A vertical window that can be shown on either the left or right of the user interface. Custom task panes can be created and added to the UI. Task panes are a convenient alternative to pop-up dialogues for editing specialized information, because they do not obstruct the map and do not require a command button or keyboard shortcut to launch them. However, this means that an Add-in that uses task panes to edit custom information must refresh when the active document and/or selection are changed.

Topic: a branch in a map. Topics can be floating topics, callout topics, or regular topics that belong to the tree. There is always one central topic in every map, which provides the starting point for hierarchical access into the tree. Callout topics belong to the parent topic to which they are attached. Floating topics are effectively the callouts of the central topic, but belong to the document rather than the central topic.

Topic Label: a text field that is used to name a topic, and is user-editable. The term "label" is also used internally when referencing the text description of a map marker.

Topic Properties: from Mindjet 11 onwards, the new name for **Custom Properties**

Transaction: a session in which Mindjet 11 suppresses display updates in the active document. This is managed by creating a transaction object, which has a callback that invokes the processing code. While this callback is running, Mindjet 11 does not make any updates to the displayed map. Transactions can also be undone in the Undo menu in a single step, so where a series of edits to a map is carried out through the API, and a partial undo of these steps would risk data integrity, a Transaction should be used.

Version: the version number of MindManager or Mindjet software consists of three parts: the major version number, the service pack number, and the build number. For example, version 10.1.459 means MindManager 10 (2012), service pack 1 (the second release), and build 459. The first number can be used to identify the various MindManager releases from 5 upwards. (MindManager 2012 is version 10).

WinWrap Basic: the macro language used by Mindjet 11 onwards.

4.2 Working with Mindjet Files

Mindjet Files is the cloud-based storage and file management system within the Mindjet system. Mindjet 11 for Windows from version 11.2 onwards contains an API for Mindjet Files. This API is not an API to the Mindjet Files server, but exposes the Desktop client's internal model reflecting the state of the currently connected server. In order to ensure that the local data is accurate, it must be **refreshed** when required, so that any changes made by other users logged into shared Mindjet Files workspaces will be reflected locally.

The API consists of a number of object classes, which all have names beginning with "MjService." In the description below, specific class or object names are used where relevant, and the *overall* API is referred to as the **MjService** API, although there is no single object or class called "MjService."

Currently there is no programmatic access to Mindjet Tasks.

Terminology

The naming of object model entities in the MjService API reflects the names used in the Mindjet Files UI:

- **Account** - a password-protected area, with Users as members, which provides access to one or more top-level folders.
- **User** - a member of an Account. Users may be owners or guests in an Account.
- **Permissions** - the actions that a user is allowed to take within an Account.
- **Folder** - a container for files and other folders.
- **File** - a data file that can be uploaded, downloaded, checked in, checked out and versioned. Files may or may not be maps. Deleted files move to the trash, and only finally disappear when permanently deleted. Certain operations on folders and files may be **Vetoed** by the server depending on current status, account limitations and user permissions.
- **Version** or **FileVersion** - an archived copy of a file at a specific date with a meta description.
- **Check-out** - locking a file so that it can be edited by one user, and becomes read-only to other users in the account.
- **Check-in** - Releasing the lock on a file. A checked-out file can also have the lock forcibly broken.

Capabilities

The MjService API can do most of what is possible through the Mindjet Files user interface, including:

- Connecting to Mindjet Files
- Managing folders
- Managing files, including check-out and check-in
- Managing file versions

The management of users, permissions and sharing is not supported through the MjService API.

Security

The Options dialog from version 11.2 onwards includes a setting to enable or disable access to the Mindjet Files features by extension software. Users can choose to prevent programmatic access to their Mindjet Files accounts by disabling the API.

The MjServiceAPI is not able to retrieve user passwords, so extension add-ins or scripts cannot be used to breach security. Locally stored or locally entered passwords can be passed with calls to access Mindjet Files accounts, but the password not exposed as a property of an account. For security reasons, your extension software should either invite users to log in to Mindjet Files through the existing UI, or should allow users to opt in to local storage of passwords. If the user allows the software to store passwords, they should be stored in an adequately encrypted form. Account passwords should not be stored in plain text (e.g. attributes or custom properties in maps, XML files or registry key values), as these are browsable.

Enabling and disabling the API

The API can be enabled and disabled by the user under File > Options > Addins. If the API is disabled, then any calls to the properties will result in an error being raised. Before using the API in your code, always check that it is enabled with the **Application.Options.MjServiceApiEnabled ('MjServiceApiEnabled Property' in the on-line documentation)** property. Note that if Mindjet Files is off-line you can still access cached files.
The **Application.Options.MjServiceEnabled ('MjServiceEnabled Property' in the on-line documentation)** property indicates whether Mindjet Files is enabled in general, as it may be disabled at registry level by an Administrative install.

MjService API Entry Point

While there are other properties elsewhere in the object model that relate to the MjService API, the main point of entry is the **Application.MjServiceClient ('MjServiceClient Object' in the on-line documentation)** property. This object is automatically instantiated and provides methods and properties for accessing Mindjet Files status and objects:

- Registration of events that fire on specific objects, e.g. monitoring a particular folder or file
- Retrieve folders, files, versions (version-controlled copies of older files) and other objects such as accounts and users by their object ID
- Retrieve the collection of Accounts to which the logged-in user has access (**Accounts ('Accounts Property' in the on-line documentation)** property)
- Retrieve the collection of top-level folders belonging to an account (**Folders ('Folders Property' in the on-line documentation)** property)
- Manage some of the Mindjet Files options, e.g. the option for automatic sign-in
- Log in and log out
- Work online or offline
- Manage the connection state that exists between the client and the remote Mindjet Files server. The event **ConnectionStateChanged ('ConnectionStateChanged Event' in the on-line documentation)** fires if there is a change in connection state. If the client loses the connection with the server, or if the user manually connects, this event will fire.

Asynchronous operations

The MjServiceAPI object acts as a cache between the client and the remote Mindjet Files server. Accessing locally stored objects is instantaneous, and many of the methods and properties return immediate values. But ensuring that the locally cached data remains *synchronized* with the remote server may take much longer, or might even time out altogether if there are communications problems. So operations that change the status on the server, or require the locally stored data to be refreshed (synchronized with the server) are handled **asynchronously**.

When an asynchronous command is issued, the method returns a **MjServiceRequest ('MjServiceRequest Object' in the on-line documentation)** object, which raises a **RequestComplete ('RequestComplete Event' in the on-line documentation)** event at a later time when the operation completes. This means that your code is not blocked while waiting for communications to complete or time out. Provided that you save the returned MjServiceRequest pointer, and handle the events that it raises, you can easily relate a later reply to a pending request. Asynchronous commands include:

- Refreshing any object or collection of objects
- Creating, deleting, renaming or copying folders or files
- Uploading and downloading files
- Checking files in or out

The process for refreshing an object is:

1. Check whether the object requires refresh by reading the **IsRefreshNeeded ('IsRefreshNeeded Property' in the on-line documentation)** property, available on all object types. Do not refresh unless a refresh is required, as this will place unnecessary load on the server. Objects only require refreshing when they are out of date with respect to the server, or if they have not yet been retrieved.
2. Invoke the **Refresh ('Refresh Method' in the on-line documentation)** method for the object.
3. Save the pointer to the MjServiceRequest object returned by the Refresh method.
4. Listen for the RequestComplete event to fire on this MjServiceRequest object. While listening, the host thread (that is waiting for the refresh) can sleep, pending the arrival of the RequestComplete event. In the macro samples included in this documentation, the host code simply waits for a flag to be set. In Add-in code, you should not block the processor this way but should allow other processes to run.
5. When the RequestComplete event fires, check the returned success flag and unblock the host thread so that it can continue.

Object IDs

All objects within the Mindjet Files system have unique and persistent IDs, represented as GUIDs in formatted text, e.g. "48483c1f-8bd1-47b3-b935-fdb356348399." This enables objects to be tracked between sessions by storing their IDs, and to be directly addressed with hyperlinks. Objects can be retrieved by ID without needing to traverse folders or account lists to find them. The **Application.MjServiceUtilities ('MjServiceUtilities Object' in the on-line documentation)** object has methods for validating that Object IDs are correctly formatted, and converting IDs directly to or from URLs. The objects that are accessible by ID are:

- Account (**MjServiceAccount ('MjServiceAccount Object' in the on-line documentation)**)
- File (**MjServiceFile ('MjServiceFile Object' in the on-line documentation)**)
- Folder (**MjServiceFolder ('MjServiceFolder Object' in the on-line documentation)**)
- Version (**MjServiceFileVersion ('MjServiceFileVersion Object' in the on-line documentation)**)
- User (**MjServiceUser ('MjServiceUser Object' in the on-line documentation)**)

When you get an MjService object from the local data model, it may be out of date with respect to the server. Most objects, including the collections of files and folders, support the **IsRefreshNeeded ('IsRefreshNeeded Property' in the on-line documentation)** property and the **Refresh ('Refresh Method' in the on-line documentation)** method, which can be called if a refresh is needed. It is more efficient to refresh a collection of objects than to individually refresh each member.

Additional document properties

The MjService API adds some new properties to the **Document ('Document Object' in the on-line documentation)** object so that Mindjet Files documents can be accurately handled. These are:

- **Document.IsMjServiceDocument ('IsMjServiceDocument Property' in the on-line documentation)** - true if the map is a document from Mindjet Files
- **Document.IsMjServiceVersion ('IsMjServiceVersion Property' in the on-line documentation)** - true if the map is a file version from Mindjet Files
- **Document.MjServiceID ('MjServiceID Property' in the on-line documentation)** - the ID of this document if loaded from

Tracking changes by listening for Events

If you need to detect and respond to changes in objects (such as a file being updated), you can listen for events with **MjServiceClient.AddObjectEvent ('AddObjectEvent Method' in the on-line documentation)** to monitor a generic type of event, or **MjServiceClient.AddObjectEventWithSource ('AddObjectEventWithSource Method' in the on-line**

documentation) to point out a specific object that you want to monitor. An object can be a folder, a file, folder collection or a file collection.

Events are issued only when an object changes from the Client's perspective. There is no continual monitoring of server activity to automatically detect changes on the server. Changes that originate from the server will only be detected when the changed object is next refreshed. To monitor an object for changes, it must be periodically checked to see if a refresh is needed with the **IsRefreshNeeded ('IsRefreshNeeded Property' in the on-line documentation)** property. If a refresh is required then it should be refreshed. When an object is changed through the GUI or API, the change event will fire immediately (before the server confirms the change.) If the change fails, another change event will be sent to the API when the object is reverted to its previous state.

Note that events are not escalated upwards in the object hierarchy. For example, if you monitor a file collection for changes, no change will be notified if one of the files in the collection changes (such as a file rename). A change event will only be issued if the collection itself changes size.

Error Conditions

The MjService API raises errors under the following conditions:

- If any operation is attempted when the MjService API is disabled or Mindjet Files is disabled: check **Application.Options.MjServiceEnabled ('MjServiceEnabled Property' in the on-line documentation)** and **Application.Options.MjServiceApiEnabled ('MjServiceApiEnabled Property' in the on-line documentation)** properties before accessing the MjService API
- If there is no connection to the Mindjet Files server
- If a login already exists, but different credentials are required: **log out ('LogOut Method' in the on-line documentation)** first before **logging in ('LogIn Method' in the on-line documentation)** again
- If a command is vetoed, e.g. trying to check out a file that is already checked out: check the **veto flags ('Vetoes Property' in the on-line documentation)** before calling a command that could be blocked
- If an invalid object ID is submitted: validate the correct format of object IDs with **MjServiceUtilities.IsValidID ('IsValidID Method' in the on-line documentation)** before use.

Code Samples

Several of the objects in the object model documentation contain sample Macros that show how to access the MjService API. Because the objects and actvities in this API are so closely related, there are a small number of longer samples that each cover several objects, methods, properties and events. They are:

- **MjServiceClient ('MjServiceClient Object' in the on-line documentation)** - shows how to establish that a valid connection exists, and how to refresh and traverse the major objects (folders, files and file versions). The example generates a map of the contents of the currently connected Mindjet Files accounts. This example is a good place to start, as the map that it generates can easily be related to the contents of the Mindjet Files browser.
- **MjServiceClient.AddObjectEvent ('AddObjectEvent Method' in the on-line documentation)** - shows how to create event objects that listen for changes in the contents of the Mindjet Files server. The example covers three cases: listening for changes across all files and folders, listening for changes in the Files collection of a specific folder, and listening for changes in a specific file. The latter two use the method **MjServiceClient.AddObjectEventWithSource ('AddObjectEventWithSource Method' in the on-line documentation)**.
- **MjServiceFolders ('MjServiceFolders Collection' in the on-line documentation)** - shows how to refresh the top-level folders collection.
- **MjServiceFolder.CreateMap ('CreateMap Method' in the on-line documentation)** - shows how to create and open a new map in a MjService Folder.

4.3 Automation Exclusions

Not all Mindjet 11 features can be accessed or controlled through the API, including:

- Built-in Mindjet 11 commands including import and export functions

- Rendering of custom data in the standard exports. For example, you cannot add the export of a custom field into the export to Word.
- Stored topic filters
- Topic alerts
- Map styles
- Controlling the viewing of map artifacts by type, as configured in the **View > Show/Hide** menu in the UI. For example, you cannot control the visibility of Floating Topics through the API.
- Hyperlinks, tables or text decoration in Topic Notes
- Spell checking and auto correction
- Topic auto numbering
- Topic sorting
- Gantt chart functionality
- Presentation mode
- Mindjet Connect functionality in Mindjet 11
- Topic auto-calculation of properties in Mindjet 11
- The Help system

5 Mindjet 11 Macros

5.1 Macro Overview

Mindjet 11 contains an embedded "macro" system, using a VBA-compatible scripting language implemented with WinWrap Basic. The macro editor & debugger are shipped with Mindjet 11 and can be accessed through the Mindjet 11 interface. This provides a quick and relatively simple way to explore the Mindjet 11 API without an external development environment, although it does have some disadvantages when compared with fully integrated Add-Ins.

The term "macro" has come to mean two distinct things in relation to software automation. Originally, a "macro" could be learned by a program by recording actions taken by the user. A particular sequence of commands or actions could be recorded and replayed, to save time with repetitive operations. This actually worked by converting actions in the user interface to lines of code that automated the application. More recently, "macro" has come to refer to a simple programming environment that does not require compilation or configuration. Microsoft Office applications support VBA (Visual Basic for Applications) to provide access to the internals of the application and the data contained in documents. Mindjet 11's macro environment provides this access, but does not have the capability to learn and repeat actions in the user interface. The only way to automate Mindjet 11 is to control it through the API by writing code.

The advantages of using macros to automate Mindjet 11 include:

- A simple programming language that does not normally need extra configuration or additional libraries.
- Easy to debug at runtime, even on target systems, resulting in very quick iterative development.
- Automatic release of object pointers when the code goes out of scope, making code inherently robust.

The disadvantages of the macro feature for developing more sophisticated solutions include:

- Limited use of some Mindjet 11 features: persistent objects cannot be used, as they can only exist while the code is in scope, i.e. while it is running. For example, to use a custom Ribbon Tab requires creating the tab dynamically and maintaining its object pointer until it is no longer required. Because the macro environment sheds all pointers when execution stops, a custom ribbon tab cannot remain in existence between successive runs of the code.
- Limited support for events, for the same reason as above: events can only be handled when the code is in scope. When the macro stops, there is nothing running that can handle an event.
- Relatively slow performance compared to an Add-in, as the code is compiled on demand.
- Lack of sophistication in areas such as file handling and user interfaces, although object oriented programming and external Windows libraries can be used.
- No protection for source code, although obfuscation can be used as a mild deterrent.
- Making macros shippable and installable requires a third party installation packager, such as Inno Setup.
- Using break points at run time works poorly inside event handlers such as transaction callbacks, often requiring Mindjet 11 to be restarted to regain control.

Migration from Sax Basic to WinWrap Basic

From Mindjet 11, the former Sax® Basic macro scripting feature has been updated to **WinWrap® Basic**. WinWrap Basic is compatible with Sax Basic, so existing scripts should work, subject to the advisory below and any other changes in the object model at Mindjet 11 that may affect your code. WinWrap Basic offers three language compatibility modes: WWB-COM, WWB.Net and WWB.Net/Compiled. Only WWB-COM is licensed in Mindjet 11, and the other two language compatibility modes cannot be used. If the WWB-COM language is not declared, then it is assumed.

Advisory (Mindjet 11)

Older Sax Basic files containing the metadata line "Attribute VB_Name = ..." may not be executable in WinWrap Basic. The symptom is that the file can be loaded, but the **Macro > Run** command, the "Start/Run" button and the **F5** key are all disabled, preventing launch of the macro. This metadata line is not normally visible in either the Sax Basic or WinWrap Basic IDEs. To work around this, open the .mmbas file with Notepad and delete the line starting "Attribute VB_Name = ".

5.2 Macro Application Areas

The technical features of the Mindjet 11 macro environment mean that it is better for some kinds of Mindjet 11 application than others.

Use macros for:

- Learning the API by experimenting with it and testing ideas
- Creating one-time tools for special processing that would be time-consuming or impractical through the user interface, e.g. to systematically modify a large number of hyperlinks in the map
- Building prototypes to prove a concept before investing in an Add-in
- Working with trusted users who do not object to launching macros to carry out special actions

By contrast, Add-ins are better than macros for:

- Creating a fully integrated solution that takes advantage of ribbon buttons, task panes or events
- Creating a solution that can be installed and uninstalled like a regular Windows application
- Applications that use Microsoft .Net technologies
- Applications that demand higher performance
- Applications that require more sophisticated forms and controls, and which need to have a consistent look & feel with Mindjet 11

The remainder of this section focuses on using the macro feature as a vehicle for learning the API and creating one-off tools or prototypes.

5.3 Macro Tutorial

If you have not used Mindjet 11's macro feature before, try the following worked example.

1. If you are using MindManager 9, ensure that the Developer tab is visible by enabling the option in **File** > **Options** > **View** page > **Developer Ribbon Tab**
2. Go to the **Developer** tab (MindManager 9) or the **View** tab (MindManager 10 and Mindjet 11)
3. Click **Macros** > **Macro Editor**. The WinWrap Basic Editor window will open.
4. Copy and paste the following code into the new empty macro document, replacing the existing "Sub Main ... End Sub":

   Macro

   ```
   Option Explicit

   Sub Main
    With VisibleDocuments.Add
      .CentralTopic.Text = "Macro tutorial"
    End With
   End Sub
   ```

5. Now click the button with the green arrow on it (Start/Resume). A new map should be created in Mindjet 11, with the central topic text "Macro tutorial".

You have now created a new macro, which starts a new Mindjet 11 map and changes the central topic text. The explanation of the code is as follows:

- "Option Explicit": this directive tells WinWrap Basic that you will be declaring all named variables with their type before using them. If you do not include this directive, then the code will allow you to use variants without saying what data type they are. Any gains from skipping the declaration of a variable are quickly lost when it comes to debugging, as you might accidentally use an inconsistent name for what should be the same variable, which can be very hard to spot. Also, you will not benefit from Intellisense (automatic suggestion of methods and properties) when working with Mindjet 11 objects, because their type will not be known in advance - they will be late bound. "Option Explicit" is usually recommended for all scripts.

- "Sub Main" ... "End Sub" declares a subroutine called *Main*, which is where execution always begins. There must be a Sub Main somewhere in every script. It does not need to be the first piece of code.
- "With **VisibleDocuments ('VisibleDocuments Property' in the on-line documentation).Add ('Add Method' in the on-line documentation)**" ... "End With": the expression "Documents(False).Add" adds a new map to the visible documents collection (which excludes hidden documents), and returns a pointer to the new **Document ('Document Object' in the on-line documentation)** object. The With ... End With construct means that everything between these lines has the context of the new document object, so we don't need to assign it to a variable and reference it.
- "**CentralTopic ('CentralTopic Property' in the on-line documentation).Text ('Text Property' in the on-line documentation)** = "Macro tutorial"": CentralTopic is a property of the document object, and we are already in the document context with the With... construct. The **CentralTopic** property returns the topic pointer of the topic that is at the centre of the map. All maps have one and only one Central Topic, so we know that by creating a new document, it is guaranteed to contain a central topic. Topics have a **Text** property which accesses a plain text version of the text held in the topic, and assigning a new value to this property replaces that text.
- When the script finishes, it cleans up any temporary pointers before going out of context. The newly created map is left in the user interface in Mindjet 11, since we did not dispose of it.

Click on the object, methods and property keyword links above to get an idea of where these entities are within the overall object model, and the other entities in the same area. For example, the **Text** property of a **Topic ('Topic Object' in the on-line documentation)** is only one of many properties.

Syntax errors and debugging

Now change ".CentralTopic.Text" to ".CentralTopc.Text". When you click **Run**, the macro stops with this line highlighted, and the message "Method or property not found" in the status bar at the bottom of the macro editor window. Click **Macro** > **Check Syntax** to check the syntax of a macro before running it. It will find and stop on the *first* error only, so you may need to repeat this step several times if your code contains multiple errors. If there are no errors, the message in the status box at the bottom of the window is "Idle".

1. Correct the above syntax error, then click once in the dark gray column to the left of the line beginning ".CentralTopic.Text = ..." A brown dot will appear in the column, and the line will be highlighted. This sets a *breakpoint* on this line of code, so the macro will pause when it gets there. Click **Run** again, and this time the script will stop on this line, highlighted in yellow. The Debugging windows will also be visible above the code, with the **Immediate** tab selected. In this debugging window, type

   ```
   ?.CentralTopic.Text
   ```

   and press Enter. As you press the period character ".", Intellisense will offer relevant suggestions for the available methods and properties. Note that you have set a breakpoint inside the *With... End With* section, so you are already in the context of a Document object. When you press Enter, the macro editor evaluates the input statement and prints the result. This is the current value of the central topic Text property, *before* execution of the line containing the breakpoint, so is the default text for a newly created map. Breakpoints stop the code before executing the line containing the breakpoint. Click the green arrow **Start/Resume** button to complete the macro, then click **Debug** > **Clear All Breaks** to unset the breakpoint again.
2. With the macro stopped (not running), click View > Debugging Windows. This makes the debugging windows visible all the time, even when the macro has finished. By using Debug.Clear and Debug.Print statements in your code, you can capture useful debugging information as the macro runs. Try adding the following line before and after the line ".CentralTopic.Text = ...":

   ```
   Debug.Print .CentralTopic.Text
   ```

   When you run the macro, you will be able to see the central topic text before and after it is changed, without using a breakpoint. Although this example is trivial, this is a useful technique for revealing internal status information to aid debugging. Many of the code samples in this documentation use Debug.Print to show output from the code.

Challenges

See if you can work out how to do the following. Suggested solutions are in the example below.

1. Before creating a new map, display the number of unsaved documents already open in Mindjet 11. Hint: the **Documents ('Documents Collection' in the on-line documentation)** collection can be iterated, and a **Document ('Documents Collection' in the on-line documentation)** object has a property that shows whether it has been modified since it was created or last saved.
2. Try adding a new Main Topic to the newly created map. Hint: a **Topic ('Topic Object' in the on-line documentation)** (of which the Central Topic is an instance) has an **AllSubTopics ('AllSubTopics Property' in the on-line documentation)** property, which returns a **Topics ('Topics Collection' in the on-line documentation)** collection. You can add a subtopic to a topic by adding it to the AllSubTopics collection, with the **Add ('Add Method' in the on-line documentation)** method of the Topics collection.
3. Try setting the 0% Task Complete icon on the new topic you just added. Hint: it will be useful to keep copy of the newly created topic pointer. Declare it first, as a Topic object. The Topic object has a **Task ('Task Property' in the on-line documentation)** property, which in turn has a **Complete ('Complete Property' in the on-line documentation)** property. Normally this is -1 (no task), but writing 0 to this property turns the topic into an unstarted task.
4. Browse the Object Model in this documentation, locate the Document object and find the method that closes a document. Then add this to your macro so that the document is not left open when the macro quits. Note that without saving the document, closing it will discard changes.

Suggested solutions

Suggested Solutions

```
Option Explicit
Sub Main
 Dim m_Document As Document
 Debug.Clear
 For Each m_Document In Documents
  If m_Document.IsModified Then
   Debug.Print "Unsaved: " & m_Document.Name
  End If
 Next
 With VisibleDocuments.Add
  .CentralTopic.Text = "Macro tutorial"
  Dim m_MainTopic As Topic
  Set m_MainTopic = .CentralTopic.AllSubTopics.Add
  m_MainTopic.Task.Complete = 0
  .Close
 End With
End Sub
```

5.4 Macro IDE

In addition to the debugging features mentioned in the Macro Tutorial, the WinWrap Basic editor also has the following features to help with macro development:

- Multi-document interface - you can have multiple macros open at the same time, and use breakpoints in related macro files.
- **Edit** > **References** allows you to create references to other DLLs, such as the Windows XML library or the Windows Script Host (WScript) to provide better file handling and registry functions.
- **Debug** > **Object Browser** allows you to browse the MindManager object model and insert keywords into your code

- The Intellisense feature automatically offers appropriate objects, methods, properties and enumerations when working with declared types

5.5 Macro Troubleshooting

Macros are easier to troubleshoot than Add-ins, because you can always use breakpoints, even when running on the target system.

Tip: breakpoints in an event handler such as a Transaction.Execute handler can prevent further events from running. You may successfully hit the first breakpoint inside an event, but when you stop and restart the macro, the event may not fire again. The workaround is to close and restart Mindjet 11. A solution for this is to split out the code that handles the event into a subroutine that is called by the event handler. You can then implement a "Debug" flag that will either call this routine in-line without starting a transaction, or will create a transaction and launch it, which then calls the same routine. Once the code has been debugged, the Debug flag can be switched off so that it runs inside a transaction handler instead, without needing any more debugging with breakpoints. You can use the same flag to manage the display of diagnostic details with Debug.Print.

Always trap runtime errors in your code. Apart from typical run-time errors such as integer overflow or out of bounds pointers, errors are also raised by Cancel buttons on dialogues, and there are many cases where Mindjet 11 will raise an error in response to what might be considered to be a normal condition. For example, if you try to remove an icon marker from a topic that does not have that marker, then instead of ignoring the request (as the condition is already satisfied), Mindjet 11 raises an error. So you can either test for the presence of the icon before deleting it, or you can delete it unconditionally and discard any resulting error. WinWrap Basic offers the choice of jumping to an error handler, or continuing past the error and checking whether Err.Code is non-zero after an operation where an error is possible. If you do not explicitly trap errors or set your code to automatically resume after an error, then the macro will stop anyway with a highlighted line and an error message in the status bar.

If you encounter the situation where Mindjet 11 refuses to run any macros at all, and throws syntax errors on MindManager objects, then run a repair on the Mindjet 11 installation to ensure that the MindManager type library is properly registered.

WinWrap Basic tip: if you use a subroutine with more than one argument, it must be invoked with "Call". Subroutines with only one argument can be invoked by simply stating their name, but if they have more than one argument then they must be invoked by prefixing this with "Call". The syntax error displayed is cryptic: if you omit the Call keyword, the macro editor says it is expecting a closing bracket after the first argument.

5.6 Macro Integration and Deployment

The MindManager Application object is automatically integrated into the macro environment, and does not need to be explicitly referenced. Unlike Add-ins, you do not need to prefix any references to MindManager objects, properties, methods or enumerations with the MindManager application itself. This context is always present. So in macro code, you can write

```
Set m_Document = ActiveDocument
```

whereas in an Add-in, you would need to write

```
Set m_Document = MindManagerApp.ActiveDocument
```

where "MindManagerApp" is the MindManager Application object passed when the Add-in is first connected.

A Mindjet 11 macro is a standalone text file with the extension .MMBas. The file can be viewed and edited in any text editor, although the built-in macro editor window is the usual way to edit. If viewed as a text file in other applications, you will see the external references (e.g. to Windows libraries) declared at the start of the file. This information is normally suppressed in the macro editor window.

Most macros are a single file, but you can create macro file sets by referring to other macro files with the syntax

```
'#Uses "another macro file.mmbas"
```

Note that this line appears as a comment in the code, but is interpreted by WinWrap Basic. Relative file paths are assumed to be relative to the current file. In a file set, only one of them may contain `Sub Main()`, and the other files should be classes or code modules.

If you have more than one version of Mindjet 11 installed, the latest version to be installed will own the .MMBas file extension by default. You can right-click on a .MMBas file in Windows Explorer and select the required version of Mindjet 11 if there is any ambiguity.

To send a macro to someone else, you only need to send them the MMBas file or file set. No other libraries or configuration files are required.

Launching macros

macros can be launched in three ways:

1. By opening the MMBas file either from Windows Explorer or from the **File** > **Open** menu in the macro editor, then running it from there with **F5** or **Macro** > **Run**.
2. By creating a registry key for the macro in **HKLM** or **HKCU** > **Software\Mindjet\MindManager\11\Macro** (where **11** is the major version number), containing the values Name, Description, Path (the path to the MMBas file) and DWORD **Menu** = 0. This causes the macro to be listed in the Macro Organizer dialog, from where it can be launched without using the editor.
3. By creating a similar registry key but assigning the macro to a dynamic command menu in Mindjet 11. With the advent of the Microsoft ribbon from MindManager 7 onwards, the number of menus to which macros can be attached has greatly diminished, as the command groups in ribbon tabs are no longer extensible menus. However, the context menus are still available.

The following example will attach the MMBas file "redtopic.mmbas" to the Topic Context menu, displaying a new command in the pop-up menu when right-clicking on a topic in MindManager 9:

Registry key: HKCU\Software\MindManager\9\Macro\RedTopic

Contains values:

(SZ) **Description** = "Turn topic text red"

(SZ) **Name** = "Red Text"

(SZ) **Path** = "...installed path...\redtopic.mmbas"

(DWORD) **Menu** = 17

The file redtopic.mmbas contains the code

```
Option Explicit

Sub Main
 Dim m_Topic As Topic
 For Each m_Topic In ActiveDocument.Selection
  m_Topic.TextColor.SetARGB(255, 255, 0, 0)
 Next
End Sub
```

For the complete list of codes that can be used in the **Menu** value for registry keys under Macro, refer to the **MmDynamicMenu ('MmDynamicMenu Enumeration' in the on-line documentation)** enumeration.

5.7 Best Practices with Macros

- Good programming principles apply to macro development as much as they do elsewhere. The key to legible programming is a consistent, accurate and descriptive naming convention for data (variables and constants). Ideally, the name for a variable should indicate its data type, its scope and its content. This convention will speed up troubleshooting.

- Always use "Option Explicit" at the start of your code, to force the interpreter to raise an error if you use a variable without declaring it. This avoids the common error where a variable is incorrectly referred to by misspelling its name. Without Option Explicit enabled, these errors can be hard to spot.
- Declaring variables as specific data types (e.g. String) instead of a general purpose Variant will improve execution time by eliminating unnecessary conversions.
- Using With... End With blocks can lighten up code, making it more readable. With... End With blocks can be nested.
- When modifying maps, use a **Transaction ('Transaction Object' in the on-line documentation)** object unless the updates really are trivial. This will considerably improve performance, as updates to the map in the user interface will be suppressed until the transaction is complete. However, remember that Task Panes which dynamically respond to maps or map content will still be updated, and other Add-ins will receive and process events from changes made through the API.
- Suppressing the Undo queue for automatically generated updates to the map can be helpful in terms of memory consumption and practicality for the user. In many cases, it does not make sense for the user to undo part of a series of actions carried out through the API. Queuing to the Undo queue can be suppressed by clearing the **IsUndoable ('IsUndoable Property' in the on-line documentation)** property of **Command ('Command Object' in the on-line documentation)** objects.
- Generally, For Each... Next loops are faster than iterating over collections with an index.
- If items are added to or deleted from a collection while iterating over it, the iteration will need to be restarted. A better way to delete all the items in a collection is to create a While... Wend loop that deletes the first item in the collection while the collection count is greater than zero.
- It is not strictly necessary to release MindManager object pointers (e.g. a Topic pointer) when you have finished using them. The macro runtime environment will take care of doing this automatically. However, if you intend to port your macro code to an Add-in, it is worth getting into the habit of releasing declared pointers, because this will be mandatory in an Add-in. Releasing pointers should be a question of habitual coding style. Finding an unreleased pointer in an Add-in that is preventing the MindManager.exe process from closing can be a tedious process, and it is better to avoid this problem in the first place.

6 Mindjet 11 Add-ins

6.1 Overview

An Add-in is a piece of code that is loaded by Mindjet 11 into the application itself, and works from within Mindjet 11. Typically, Add-ins create new ribbons, command buttons, or task panes, and add more features and functions in a seamlessly integrated way, so that they work together with existing Mindjet 11 functions. Several of Mindjet 11's factory-shipped features are implemented as Add-ins.

Although Add-ins work inside Mindjet 11, they do not become part of the core code. They access Mindjet 11 through an API, provided as a COM server. On the other side of this interface, Mindjet 11 validates and sanitizes the requests that are made, so Add-ins do not directly access Mindjet 11 's memory or code. Add-in events also run in their own process threads, so to some extent the misbehavior of an Add-in can be tolerated without compromising Mindjet 11 or making it unresponsive.

Add-ins can be installed in the same way as regular programs, and can have associated resources such as files, documentation, template maps and so on. The main difference between installing a standalone program and an Add-in for an existing program is that in the case of the Add-in, no entry is created in the Start menu, and no shortcut on the Windows desktop is created. The Add-in cannot be independently launched, but gets launched only when Mindjet 11 starts.

Add-ins take longer to develop than macros, but are more suitable for applications that need better performance, better security and require persistent user interface items such as ribbons. While an Add-in remains loaded, it can keep pointers open to UI customizations, meaning that they stay in scope even when the Add-in code itself is not being executed.

6.2 How Add-ins Work

Mindjet 11 for Windows Add-ins use the same technology as Microsoft® Office® extensions. Technically, an Add-in is typically comprised of:

- Configuration data in the system registry that tells Mindjet 11 that the Add-in is installed, and whether to load the DLL at start up,
- Configuration data in the system registry that identifies this DLL as a COM Automation server,
- A DLL file on disk, and
- Optional run-time resources for the DLL (such as associated images or configuration data) on disk.

Registry configuration

Data in the system Registry tells Mindjet 11 how and when to load Add-ins. This data is read once at Mindjet 11 start-up, and again if an Add-in is newly configured for loading in the Add-ins page in the Mindjet 11 Options dialog. The primary source of data is in HKLM under Software/Mindjet/MindManager/11/Addins (where "11" is the major version), which contains a key for each installed Add-in. The key is usually named after the Add-in Connect class, e.g. "MyAddIn.Connect", and contains the friendly name and description as text, and a DWORD value called "LoadBehavior". If the LoadBehavior value is set to 2, then Mindjet 11 will attempt to load the Add-in when it starts up. It will create a similarly named key and value in HKCU for the current user, which reflects the individual load preferences and the current state of loading.

To locate the DLL file for the Add-in, Mindjet 11 looks up the CLSID for the Add-in as registered under HKCR/MyAddIn where "MyAddIn" is the name of the Add-in as declared in Mindjet 11's HKLM key. From there, it finds the CLSID (as a GUID) and looks that up under HKCR/CLSID/{clsid}. This gives access to the different versions of the Add-in that have been registered, and the name and location of the DLL file on disk. Mindjet 11 will usually load the *newest* version of the DLL; if you inadvertently build an older version than one already registered, then it will not get loaded.
The Windows utility **regsvr32** creates the entries in HKCR and allows Mindjet 11 to instantiate a COM server by knowing the name of the server. Normally, this registration is carried out when the Add-in is installed. Note that Mindjet 11 looks for a DLL in the same location as the runtime, i.e. the the same location as MindManager.exe. If the DLL already exists there, then that copy will be loaded. If it does not exist there, then it will look for copy pointed

out by the entry in HKCR. This can cause confusion, especially if an Add-in under development is also *installed* on the development system, which might be undesirable.

The DLL file

The DLL file for an Add-in contains a class called "Connect" implementing a known and published Windows interface called "IDTExtensibility2". This interface supports a set of five methods with defined signatures, i.e. with predefined names and arguments. When Mindjet 11 loads this Add-in, it makes calls to these procedures at specific times. Of course, you can have lots of other code and external methods & properties in the DLL as well, but the Connect class *must* implement the IDTExtensibility2 interface and support the methods of that interface for Mindjet 11 to communicate with it. The five methods expected by Mindjet 11 are:

Connect.OnConnection() Method

The OnConnection method is called when MindManager loads and connects to the Add-In. It can be called either when Mindjet 11 starts (if the Add-in behavior is set to connect on start-up), or if the user enables a previously disabled Add-in by checking it in the Mindjet 11 Options > Add-Ins dialogue. If the Add-in is newly enabled, then the connection method is called when the user clicks OK. Therefore, OnConnection() can also be invoked when Mindjet 11 is either still initializing, or is fully up and running. The **connectMode** parameter passed with the call indicates which; at startup, connectMode is **ext_cm_Startup**, and if Mindjet 11 is already initialized, then connectMode is **ext_cm_AfterStartup**. This should be used to correctly initialize the Add-in.

Mindjet 11 also passes a pointer to its **Application ('Application Object' in the on-line documentation)** object with this method, which the Add-in should save and re-use each time it needs to access the Mindjet 11 application. This is the only time that Mindjet 11 advises an Add-in of the Application object pointer, so this pointer is typically saved in a variable with global scope inside the Add-in code. The exact state of Mindjet 11 during the loading of Add-ins at start-up is undefined. You should not assume that Mindjet 11 is fully ready to operate, or that all other Add-ins or start-up documents are loaded. If any unhandled exceptions occur during the OnConnection() method, then Mindjet 11 will not invoke any of the other methods, and will automatically flag the Add-in as disabled, preventing it from being automatically connected at the next launch. It is therefore vital that all exceptions are properly handled inside the Add-In.

Connect.OnStartupComplete() Method

The OnStartupComplete method is called by Mindjet 11 when it has completed all start-up initialization, loading of Add-Ins, and the user interface is ready. Obviously, only Add-ins that are successfully loaded and connected will have this method called. So while it would sometimes be useful to defer initialization actions until Mindjet 11 is ready to execute them, remember that this method is only called once, and is not re-issued later if an Add-in is newly connected from the Add-Ins page in the Options dialog. If your Add-in requires initialization that needs Mindjet 11 to be fully operational, then use the following strategy:

- Factor all the deferred initialization actions into a separate subroutine.
- In the OnConnection() method, call this initialization subroutine only if the connectMode argument is **ext_cm_AfterStartup**, indicating that connection is manually initiated. Mindjet 11 will be running normally at this point.
- In the OnStartupComplete() method, also call this initialization subroutine, which will initialize in the event that the Add-in was loaded at startup.

Connect.OnAddInsUpdate() Method

The OnAddinsUpdate method is called by Mindjet 11 whenever a change is made to the Add-ins configuration, through the Add-ins page in the Options dialog. Enabling or disabling any of the Add-ins will cause this method to be called in all Add-ins that are still (or newly) loaded after the change has been made. If your Add-in interacts with others, or if you need to be aware of the presence or absence of other Add-ins, then you should handle this method.

Connect.OnBeginShutdown() Method

The OnBeginShutdown method is called in all loaded Add-ins when the user initiates shutdown of the Mindjet

11 user interface. It is an early warning that disconnection will follow soon. For example, if you need to save any open map data to disk before Mindjet 11 closes, then this method would be a better place than the subsequent OnDisconnection() method, because Mindjet 11 is still operating properly at this point. When disconnection actually starts, you can no longer rely on documents or other Add-ins being present and operational.

Connect.OnDisconnection() Method

The OnDisconnection method is called just before Mindjet 11 unloads the Add-in. This can occur as part of a normal shutdown, and also if the Add-in is manually disconnected in the Add-ins page in the Mindjet 11 Options dialog. Therefore, the OnDisconnection() method does not necessarily signal that Mindjet 11 itself is closing down. The context is indicated by the **RemoveMode** argument; if a manual disconnection is taking place while Mindjet 11 is running, this will be **ext_dm_UserClosed**, and if it is occurring due to Mindjet 11 closing down, it is **ext_dm_HostShutdown**.

A pattern similar to the initialization strategy can be used if there are shutdown activities that require Mindjet 11 to be running normally when the Add-in disconnects:

- Factor the de-initialization activities into a separate subroutine.
- Call this subroutine in the OnDisconnection method only if the RemoveMode argument is **ext_dm_UserClosed**, indicating a manual disconnection without shutdown. Mindjet 11 will still be running normally at this point.
- Call this subroutine in the OnBeginShutdown() method.

During the OnDisconnection() method, the status of documents and other Add-ins is not guaranteed, so you should handle any exceptions that might arise during OnDisconnection() even though they would work normally elsewhere. All pointers to Mindjet 11 objects (including the Application object itself) should be released at disconnection, otherwise Mindjet 11 may fail to close down and the MindManager.exe process will remain in memory, even though the user interface may disappear.

Responding to other events

The above five methods cover the loading and management of Add-ins, but do not themselves provide any ongoing interaction so that the Add-in can respond to user commands or changes in Mindjet 11. These are notified by the **events** for which the Add-in registers itself as a listener, and provides a handler procedure. Examples of typical events include:

- A **Click ('Click Event' in the on-line documentation)** event from a Command object that is associated with a ribbon button or context menu item, and fires when the user clicks on the button or menu command
- An **UpdateState ('UpdateState Event' in the on-line documentation)** event from a Command object, that is periodically raised by Mindjet 11 when it wants to refresh the display of ribbon button
- A event from a configured **Event ('Event Object' in the on-line documentation)** object raised by Mindjet 11 to signal that the selected topic(s) have changed, a document has been opened, a topic has been modified and so on
- The Idle event raised periodically by Mindjet 11, to indicate that the user is drinking coffee, and Mindjet 11 is waiting for something to do
- A callback event from a **Transaction ('Transaction Object' in the on-line documentation)** object that is requesting execution

Since Add-in code remains loaded and in scope, it can maintain pointers to interface customizations (such as custom task panes, ribbons and commands), and handle events raised by the objects that they create.

The DLL requires the Mindjet 11 type library MindManager.tlb, which defines the COM interface. This file should not be copied by the Add-in installer, and the Add-in should default to using the one registered in the GAC by Mindjet 11.

Run-time resources

Several resource types can be embedded in the assembly by using the Resources facility in Visual Studio, but many Add-ins also require external resources at run time, such as images for ribbon buttons, custom icon files, configuration files or documentation files. These are typically kept in a sub-folder relative to the installed location of the DLL, and installed with the DLL. This location can be derived at run time by reading the location of the currently executing assembly.

6.3 Development Tutorial

This short tutorial will take you through the steps of creating a MindManager Add-in in Microsoft Visual Studio. Rather than provide a ready-made template, the individual steps are explained. The tutorial takes you from configuring Visual Studio through creating and debugging a new MindManager Add-in, to building an installer for it.

6.3.1 Step 1: System Configuration

The first step is to confirm that your system is correctly configured for developing Mindjet 11 Add-ins. You will need Microsoft® Visual Studio 2005, 2008 or 2010, capable of building DLLs. (Some editions may have limitations that preclude the building of DLLs). A licensed version of the target Mindjet 11 platform should also be installed on the same system. If you create Add-ins for the oldest available build of a particular major version, then they will also work on newer builds. But if you create add-ins using the latest build, then they may fail to load on older versions. For example, if you build an Add-in using MindManager 9.1, then it will work on 9.1 and 9.2. But if you build it using 9.2, then it will work on 9.2 but may fail to load on 9.1. Unless you can be sure that your users will always have access to the latest Mindjet 11 version, it is a good idea to use the oldest stable version that you can for development.

If you are developing for Mindjet 11 or later, your IDE will need to support Microsoft .Net Framework 4 (full edition, not the Client Profile version).

Note that Add-ins built for a specific major version of Mindjet 11 will not work on a different version, e.g. an Add-in for MindManager 9 will not work on MindManager 10 (2012) unless it is re-built with the MindManager 10 type library. (In fact, Add-ins from a different major version will load and will *nearly* work, but will behave strangely, and formerly well-behaved code will throw inexplicable errors. If you find yourself trying to debug something basic that has worked hundreds of times before, then check that the DLL that is loaded is actually the one that was built for this version of MindManager. There is no automatic check on the MindManager application version when loaded, so this is something that would be worth including in the Connect.OnConnection() method.)

6.3.2 Step 2: Configure Visual Studio

Microsoft Visual Studio can create "shared Add-in" projects that will register the DLL when built, enabling the host application to automatically load it. By default, Visual Studio is configured to create Add-ins for Microsoft Office® products. To create Add-ins for other hosts requires some additional configuration. This configuration creates the minimum registry keys that allow Mindjet 11 on the development system to find and load the latest version of the DLL compiled by Visual Studio. You can do this manually (by creating the necessary keys in HKCR) but it is easier to let Visual Studio do this.

To configure this, you need to create and run a registry file that adds data to the Visual Studio configuration. Copy the code below into a plain text file called "VS Wizard for MindManager.reg" and execute it to add MindManager 8, MindManager 9, MindManager 2012 and Mindjet 11 to Visual Studio 2005, 2008 and 2010.

VS Wizard for MindManager.reg

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
```

```
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
"DisplayAdvFeatures"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1b]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1b\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1b\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1b\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1b\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1c]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1c\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1c\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1c\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard7.1c\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2005]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VisualStudio\\8.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio"
```

```
"Version"="8.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2005 Macros]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VSA\\8.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio Macros"
"Version"="8.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2008]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VisualStudio\\9.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio"
"Version"="9.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2008 Macros]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VSA\\9.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio Macros"
"Version"="9.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2010]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VisualStudio\\10.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio"
"Version"="10.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Microsoft Visual
Studio 2010 Macros]
"StartupFlag"=dword:00000001
"RegistryRoot"="SOFTWARE\\Microsoft\\VSA\\10.0\\AddIns"
"XMLAppName"="Microsoft Visual Studio Macros"
"Version"="10.0"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
"Version"="8.0"
"XMLAppName"="MindManager 8"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
"Version"="9.0"
"XMLAppName"="MindManager 9"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
"Version"="9.0"
"XMLAppName"="MindManager 2012"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.0\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
"Version"="9.0"
"XMLAppName"="Mindjet 11"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.1]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.1\MindManager 8]
"RegistryRoot"="Software\\Mindjet\\MindManager\\8\\AddIns"
"DisplayAdvFeatures"=dword:00000000
```

```
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.1\MindManager 9]
"RegistryRoot"="Software\\Mindjet\\MindManager\\9\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.1\MindManager 10]
"RegistryRoot"="Software\\Mindjet\\MindManager\\10\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\AddinWizard8.1\Mindjet 11]
"RegistryRoot"="Software\\Mindjet\\MindManager\\11\\AddIns"
"DisplayAdvFeatures"=dword:00000000
"StartupFlag"=dword:00000002
```

6.3.3 Step 3: Create a New Project

With Mindjet / MindManager installed and the Visual Studio Add-in Wizard configured, the next step is to create a new project in Visual Studio.

Before doing this, it is vital to be clear about which version of application you are developing for. There are several points at which this choice must be consistent. A Mindjet/MindManager Add-in and associated installation are designed and released for a specific major version of host, and although the majority of the code will be interchangeable, some small but vital elements are not. This version of Mindjet/MindManager must also be installed on the development system. You can have more than one version of Mindjet/MindManager host installed simultaneously. In the example below, Mindjet 11 is assumed.

When the project is created, we will need to add a reference to the correct type library for the host

1. In Visual Studio, click File > New Project..., then go to Other Project Types / Extensibility / Shared Add-in
2. Enter the Solution name. Choose this name carefully, as it will be awkward to change later on. The name entered will be the name of the COM server registered in Windows, and the name of the registry key in HKLM. It must consist of legal filename characters and not contain spaces. It must be unique within the set of other Add-in names in Mindjet 11, so "NewAddin" is not a good choice. It is best to name your Add-in after your organization, the host Mindjet 11 version and the project or product name, e.g. "ABC_MM10_DataExport". This will be easy to recognize amongst the list of Add-ins at HKLM/Software/Mindjet/MindManager/X/Addins, where "X" is the major version number.
3. Choose a location and check the option to create a sub-folder for the solution if required. The above name given for the Add-in will be used for the folder name. Again, this is not simple to change later, so do not rush this step.
4. Continue to the next dialogue, which launches the Add-in Wizard for Visual Studio.
5. Select your required language for the Add-in. The examples in this tutorial show both C# and VB.Net.
6. If the configuration at the previous step completed successfully, you should have various versions of Mindjet/MindManager listed as potential host applications. By default, the Wizard will select all the Microsoft® Office® applications as well. Un-check all of the Office applications, and select only one of the hosts. In our example, we are selecting Mindjet 11.
7. The Wizard prompts for the Add-in name again. This version is for display purposes only (stored in the registry as the FriendlyName value), and is shown in the Add-ins page in the Mindjet 11 Options dialog. You do not need to choose the same name as was given when the project was created, but can choose something that will be easy for users to identify.
8. The Wizard also prompts for a description. This is stored in the registry as the Description value, and is displayed in the Add-ins page in the Mindjet 11 Options dialog, when this Add-in is selected.
9. On the next page, configure the Add-in to be loaded when Mindjet 11 starts. This sets the value of the LoadBehavior value in the registry to 2, and means that the checkbox for this Add-in in the Options dialog will be checked.
10. Configure the Add-in to be available to all users, and complete the Wizard. This configuration causes the

Setup project to create the correct keys and values in HKLM instead of HKCU. On completion, the Wizard will create a new project and a module for the Connect class, analyzed in the next step.

11. Remove the references to Microsoft Office, and add a reference to the Mindjet 11 type library.
    a. For Visual Basic, right-click on the project in the Solution Explorer pane and go to the Properties window for the DLL project. Go to the References tab, and remove the reference to Microsoft Office, which gets added by default. Click on Add, then go to the COM tab and check the box next to "MindManager 11 Type Library".
    b. For C#, browse the Solution Explorer down to References section for the DLL project. Select the Microsoft.Office.Core reference, right-click on it and delete it. Right-click on the References heading and select Add New Reference, then select the COM tab and locate the Type Library for the required version of Mindjet 11.
12. Ensure that the assembly will be COM-visible, so that Mindjet 11 can load it. Right-click on the DLL project in the Solution Explorer and select Properties.
    a. For VB.Net, go to the Assembly tab (VS 2008) or the Application tab (VS2010) in the DLL project properties and click the button "Assembly Information". Ensure that the option "Make Assembly COM-Visible" is checked.
    b. For C#, go to the Application tab in the DLL project properties and click the button "Assembly Information". Ensure that the option "Make Assembly COM-Visible" is checked.

6.3.4 Step 4: Review Connect Code

When the Visual Studio Wizard created a new Add-in project, it created a module called "Connect", containing the Connect class. The code will look similar to this:

Connect.vb

```
imports Extensibility
imports System.Runtime.InteropServices
<GuidAttribute("E87EBE41-70E5-4CFF-8EA4-EB4089DE9FDA"),
ProgIdAttribute("VBExample1.Connect")> _
Public Class Connect

Implements Extensibility.IDTExtensibility2
    Private applicationObject As Object
    Private addInInstance As Object

Public Sub OnBeginShutdown(ByRef custom As System.Array) Implements
Extensibility.IDTExtensibility2.OnBeginShutdown
End Sub

Public Sub OnAddInsUpdate(ByRef custom As System.Array) Implements
Extensibility.IDTExtensibility2.OnAddInsUpdate
End Sub

Public Sub OnStartupComplete(ByRef custom As System.Array) Implements
Extensibility.IDTExtensibility2.OnStartupComplete
End Sub

Public Sub OnDisconnection(ByVal RemoveMode As Extensibility.ext_DisconnectMode, ByRef
custom As System.Array) Implements Extensibility.IDTExtensibility2.OnDisconnection
End Sub

Public Sub OnConnection(ByVal application As Object, ByVal connectMode As
Extensibility.ext_ConnectMode, ByVal addInInst As Object, ByRef custom As System.Array)
Implements Extensibility.IDTExtensibility2.OnConnection
    applicationObject = application
    addInInstance = addInInst
```

```
End Sub
End Class
```

connect.cs

```
namespace Harport_CS_AddIn_1
{
 using System;
 using Extensibility;
 using System.Runtime.InteropServices;
 #region Read me for Add-in installation and setup information.
 // When run, the Add-in wizard prepared the registry for the Add-in.
 // At a later time, if the Add-in becomes unavailable for reasons such as:
 //   1) You moved this project to a computer other than which is was originally
created on.
 //   2) You chose 'Yes' when presented with a message asking if you wish to remove the
Add-in.
 //   3) Registry corruption.
 // you will need to re-register the Add-in by building the Harport_CS_AddIn_1Setup
project,
 // right click the project in the Solution Explorer, then choose install.
 #endregion

 /// <summary>
 ///   The object for implementing an Add-in.
 /// </summary>
 /// <seealso class='IDTExtensibility2' />
 [GuidAttribute("6A0D6381-74C4-43DB-9684-6831B4AA72E9"),
ProgId("Harport_CS_AddIn_1.Connect")]
 public class Connect : Object, Extensibility.IDTExtensibility2
 {
  /// <summary>
  ///  Implements the constructor for the Add-in object.
  ///  Place your initialization code within this method.
  /// </summary>
  public Connect()
  {
  }
  /// <summary>
  ///      Implements the OnConnection method of the IDTExtensibility2 interface.
  ///      Receives notification that the Add-in is being loaded.
  /// </summary>
  /// <param term='application'>
  ///      Root object of the host application.
  /// </param>
  /// <param term='connectMode'>
  ///      Describes how the Add-in is being loaded.
  /// </param>
  /// <param term='addInInst'>
  ///      Object representing this Add-in.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnConnection(object application, Extensibility.ext_ConnectMode
connectMode, object addInInst, ref System.Array custom)
  {
   applicationObject = application;
   addInInstance = addInInst;
  }
  /// <summary>
  ///     Implements the OnDisconnection method of the IDTExtensibility2 interface.
  ///     Receives notification that the Add-in is being unloaded.
  /// </summary>
  /// <param term='disconnectMode'>
  ///      Describes how the Add-in is being unloaded.
  /// </param>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnDisconnection(Extensibility.ext_DisconnectMode disconnectMode, ref
System.Array custom)
  {
  }
  /// <summary>
  ///      Implements the OnAddInsUpdate method of the IDTExtensibility2 interface.
  ///      Receives notification that the collection of Add-ins has changed.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnAddInsUpdate(ref System.Array custom)
  {
  }
  /// <summary>
  ///      Implements the OnStartupComplete method of the IDTExtensibility2 interface.
  ///      Receives notification that the host application has completed loading.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnStartupComplete(ref System.Array custom)
  {
  }
  /// <summary>
  ///      Implements the OnBeginShutdown method of the IDTExtensibility2 interface.
  ///      Receives notification that the host application is being unloaded.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnBeginShutdown(ref System.Array custom)
  {
  }

  private object applicationObject;
  private object addInInstance;
 }
}
```

Let's look at this code to pick out the important features.

1. A GUID is automatically assigned for this Add-in. This will remain unchanged for the lifetime of the software, and means that you can update an existing installation with a new version. Note that if you copy an Add-in project to start another Add-in, you **must** change the GUID of the copy project to something unique, otherwise you will have two add-ins with the same GUID. This would cause problems with registration of the DLL.

2. The ProgId Attribute is declared as the project name with ".Connect" appended to it. This will be the name of the registry key added to HKLM/Software/Mindjet/MindManager/X/Addins (where X is the major version number) which holds the values allowing MindManager to manage this Add-in. When you create the Setup project, you will need this name for creating registry key.
3. The public class **Connect** implements the IDTExtensibility2 interface, which is what MindManager is expecting to find so that it can interact with the DLL after loading it.
4. Stub routines for the five methods are declared. The names and signatures of these methods should not be modified, otherwise the Add-in will not implement the required interface any more. You can edit the names of the arguments passed to the method, as this does not change the signature.
5. By default, the Visual Studio Add-in Wizard for C# adds a stub for the Connect class constructor. You can perform system-level initialization in the constructor routine, but you will not yet know where the MindManager Application is, until the OnConnection call. If you want to perform system-level initialization prior to connection for a VB.Net Add-in, override the New() method.

6.3.5 Step 5: Modify Connect Code

You can now modify the code in Connect.vb to start creating a functional Add-in. In this example, we will only aspire to the usual "Hello World" level of functionality, but will show you some useful debugging techniques.

Modify the Connect.vb module as follows. If you copy and paste from the examples below, be sure not to copy and paste the Namespace, GuidAttribute or ProgIdAttribute values, as these are unique to this example only and should not overwrite the values in your own project.

Connect.vb

```
imports Extensibility
imports System.Runtime.InteropServices
#Region " Read me for Add-in installation and setup information. "
#End Region
<GuidAttribute("E87EBE41-70E5-4CFF-8EA4-EB4089DE9FDA"), _
ProgIdAttribute("VBExample1.Connect")> _
Public Class Connect
    Implements Extensibility.IDTExtensibility2
    Private Const T_AddInName As String = "VBExample1"
    Private m_MindManager As Mindjet.MindManager.Interop.Application = Nothing
    Public Sub OnBeginShutdown(ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnBeginShutdown
        p_Trace(".Connect.OnBeginShutdown()")
    End Sub
    Public Sub OnAddInsUpdate(ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnAddInsUpdate
        p_Trace(".Connect.OnAddInsUpdate()")
    End Sub
    Public Sub OnStartupComplete(ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnStartupComplete
        p_Trace(".Connect.OnStartupComplete()")
    End Sub
    Public Sub OnDisconnection(ByVal RemoveMode As Extensibility.ext_DisconnectMode, _
    ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnDisconnection
        p_Trace(String.Format(".Connect.OnDisconnection(RemoveMode = {0})",
RemoveMode.ToString))
        MsgBox("Goodbye", MsgBoxStyle.OkOnly, T_AddInName)
        ' release MindManager application pointer
        m_MindManager = Nothing
        System.GC.Collect()
    End Sub
    Public Sub OnConnection(ByVal application As Object, ByVal ConnectMode As
Extensibility.ext_ConnectMode, _
    ByVal addInInst As Object, ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnConnection
        m_MindManager = CType(application, Mindjet.MindManager.Interop.Application)
        p_Trace(String.Format(".Connect.OnConnection(ConnectMode = {0})",
ConnectMode.ToString))
        MsgBox("Hello World", MsgBoxStyle.OkOnly, T_AddInName)
    End Sub
    Private Sub p_Trace(ByVal s_Message As String)
        ' Output a time stamped trace message
        Trace.WriteLine(String.Format("[{0} {1}] {2}", _
        T_AddInName, Format(Now, "HH:mm:ss.fff"), s_Message))
    End Sub
End Class
```

connect.cs

```
namespace Harport_CS_AddIn_1
{
 using System;
    using System.Text;
 using Extensibility;
 using System.Runtime.InteropServices;

 #region Read me for Add-in installation and setup information.
 #endregion

 /// <summary>
 ///   The object for implementing an Add-in.
 /// </summary>
 /// <seealso class='IDTExtensibility2' />
 [GuidAttribute("6A0D6381-74C4-43DB-9684-6831B4AA72E9"),
ProgId("Harport_CS_AddIn_1.Connect")]
 public class Connect : Object, Extensibility.IDTExtensibility2
 {
  /// <summary>
  ///  Implements the constructor for the Add-in object.
  ///  Place your initialization code within this method.
  /// </summary>
  public Connect()
  {
  }
  /// <summary>
  ///      Implements the OnConnection method of the IDTExtensibility2 interface.
  ///      Receives notification that the Add-in is being loaded.
  /// </summary>
  /// <param term='application'>
  ///      Root object of the host application.
  /// </param>
  /// <param term='connectMode'>
  ///      Describes how the Add-in is being loaded.
  /// </param>
  /// <param term='addInInst'>
  ///      Object representing this Add-in.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnConnection(object application, Extensibility.ext_ConnectMode
connectMode, object addInInst, ref System.Array custom)
```

```
        {
            m_MindManager = (Mindjet.MindManager.Interop.Application)application;
            addInInstance = addInInst;
            p_Trace(String.Format(".Connect.OnConnection(connectMode = {0})",
connectMode.ToString()));
            System.Windows.Forms.MessageBox.Show("Hello world", T_AddInName);
  }
  /// <summary>
  ///     Implements the OnDisconnection method of the IDTExtensibility2 interface.
  ///     Receives notification that the Add-in is being unloaded.
  /// </summary>
  /// <param term='disconnectMode'>
  ///      Describes how the Add-in is being unloaded.
  /// </param>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnDisconnection(Extensibility.ext_DisconnectMode disconnectMode, ref
System.Array custom)
  {
            p_Trace(String.Format(".Connect.OnDisconnection(disconnectMode = {0})",
disconnectMode.ToString()));
            System.Windows.Forms.MessageBox.Show("Goodbye", T_AddInName);
            m_MindManager = null;
  }
  /// <summary>
  ///      Implements the OnAddInsUpdate method of the IDTExtensibility2 interface.
  ///      Receives notification that the collection of Add-ins has changed.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnAddInsUpdate(ref System.Array custom)
  {
            p_Trace(".Connect.OnAddInsUpdate()");
  }
  /// <summary>
  ///      Implements the OnStartupComplete method of the IDTExtensibility2 interface.
  ///      Receives notification that the host application has completed loading.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnStartupComplete(ref System.Array custom)
  {
            p_Trace(".Connect.OnStartupComplete()");
  }
  /// <summary>
  ///      Implements the OnBeginShutdown method of the IDTExtensibility2 interface.
  ///      Receives notification that the host application is being unloaded.
  /// </summary>
  /// <param term='custom'>
  ///      Array of parameters that are host application specific.
  /// </param>
  /// <seealso class='IDTExtensibility2' />
  public void OnBeginShutdown(ref System.Array custom)
  {
            p_Trace(".Connect.OnBeginShutdown()");
  }

        /// <summary>
        /// Output a diagnostic line to the Trace window
        /// </summary>
        /// <param name="s_Message"></param>
        private void p_Trace(string s_Message)
        {
            System.Diagnostics.Trace.WriteLine(String.Format("[{0} {1}] {2}",
T_AddInName,
            String.Format("{0:0#}:{1:0#}:{2:0#}.{3:00#}", DateTime.Now.Hour,
DateTime.Now.Minute,
            DateTime.Now.Second, DateTime.Now.Millisecond), s_Message));
        }
        private string T_AddInName = "C#Example1";
        private Mindjet.MindManager.Interop.Application m_MindManager;
        private object addInInstance;
    }
}
```

Notes

1. We declare a constant **T_AddInName** in the class for the Add-in name. This is a suggestion only, and is useful for consistently identifying the Add-in in various places.
2. We also declare a variable **m_MindManager** to hold the Application object pointer. Because we earlier added a reference to the MindManager object model, the namespace Mindjet.MindManager.Interop can be used to declare variables and constants.
3. Note that neither of these items (T_AddInName and m_MindManager) can be exposed outside this class without using a pointer for the instantiated Connect class. You might find it more convenient to declare these items with Friend scope in a code module, where they can be accessed by any module or class in the project. They are shown inside this class for illustration only.
4. In all the methods, we output a line of diagnostic text to the "Immediate" window in Visual Studio while the Add-in is running on the development platform. We will see how to access this text at the next step. This will tell us when MindManager is calling the methods of the Connect class. It is good practice to always prefix these diagnostic trace lines with the name of the Add-in, as you might find that other Add-ins also output Trace messages to the same window.
5. In the OnConnection() method, we cast the application object to be a MindManager Application, and save it. This object will be used for all accesses to the MindManager object model by the Add-in, and is not available again from any subsequent callbacks or event callbacks, so it is important to save it. We also output a messagebox announcing that the Add-in is connected and alive.
6. In the OnDisconnection() method, we dispose of the MindManager application pointer. If we did not do this, then MindManager.exe would not close, as a pointer to the application object would remain open. The symptoms of this are that the Mindjet 11 window disappears, but the MindManager.exe process is still visible in the process monitor.

We are now ready to run and debug this Add-in.

6.3.6 Step 6: Run And Debug

If there are no syntax errors visible in the code, then we are ready to debug the Add-in.

The easiest way to do this is to configure Mindjet 11 to launch the Add-in at startup, and debug it with Visual Studio. To do this:

- Right-click on the DLL project in the Solution Explorer in Visual Studio, and select **Properties**
- Ensure that the Trace option is enabled:
  - For VB.Net, select the **Compile** tab > Advanced Compilation Options and enable the **Trace** option

- For C#, select the **Build** tab and enable the Trace option
- Select the **Debug** tab and select the option to launch an external program. Browse to the executable for the version of Mindjet 11 for which you are building the Add-in. This is why Mindjet 11 must also be installed on the development system. For example, if you are developing for MindManager 2012, then the executable will typically exist at C:\Program Files\Mindjet\MindManager 10\MindManager.exe
- In the **Debug** menu, click **Windows** > **Immediate** to display the Immediate window. This will allow you to still see trace messages after execution has finished.

When you launch the Add-in from Visual Studio, the following will now happen:

- The DLL project will be built if there have been any changes to the source code since the last build
- The Mindjet 11 host will be launched
- Mindjet 11 will attempt to load the DLL just built, and will call the Connect.OnConnection() method
- The source code will be blocked for edit inside Visual Studio
- You can set breakpoints in the DLL code
- The Immediate window will be displayed, allowing you to view Trace output and query the contents of variables when a breakpoint is hit.

Starting the run

Click the green "Run" button in the Visual Studio toolbar. Mindjet 11 will start up, and you should see the "Hello World" message box from the Add-in.

In the **Immediate** window in Visual Studio, you should see the following diagnostic texts. Diagnostic texts from the VB.Net version are shown below. The C# texts will be similar.

```
[VBExample1 11:45:49.500] .Connect.OnConnection(ConnectMode = ext_cm_Startup)
[VBExample1 11:45:51.031] .Connect.OnStartupComplete()
```

Testing disconnection and reconnection

You can now test the calls to the DLL when the Add-in is disconnected. In Mindjet 11 , go to the Mindjet 11 Options dialog and select the Add-ins page. Locate your new Add-in in the list, un-check it, then click OK. This will disconnect and unload your Add-in. In the Immediate window in Visual Studio, you should see the following:

```
VBExample1 11:46:24.437] .Connect.OnDisconnection(RemoveMode = ext_dm_UserClosed)
```

This shows how Mindjet 11 disconnects a running Add-in while it is still running. Note that you do not see the OnAddInsUpdate() method, as this is only called for Add-ins that are still loaded and connected after the configuration has been changed. Return to the Mindjet 11 Options dialog and re-enable your Add-in again, which should add the following to the Immediate window:

```
[VBExample1 11:46:38.031] .Connect.OnConnection(ConnectMode = ext_cm_AfterStartup)
[VBExample1 11:46:42.953] .Connect.OnAddInsUpdate()
```

This time, you receive the call to OnAddInsUpdate(), as your Add-in has been reconnected. Note that the ConnectMode argument is different to that passed to the OnConnection() method when Mindjet 11 started up.

Testing Mindjet 11 shutdown

Now close down Mindjet 11 normally, by clicking the close button in the Mindjet 11 window. This will return control to Visual Studio, and the Immediate window will be continue to displayed because we enabled it before starting the run. You should be able to see that Mindjet 11 informed the Add-in that it was closing down, then disconnected it:

```
[VBExample1 11:46:49.656] .Connect.OnBeginShutdown()
[VBExample1 11:46:49.703] .Connect.OnDisconnection(RemoveMode = ext_dm_HostShutdown)
```

The above debug process will speed up the development of Add-ins. You can also set breakpoints in the code, but be cautious with breakpoints in time-critical events, where the breakpoint may disrupt correct operation. Sometimes it is better to output a line using the Trace statement rather than halt the code.

If your Add-in causes Mindjet 11 to become unresponsive, you can still close it by halting the execution from Visual Studio.

6.3.7 Step 7: Build An Installer

Now that we have created and debugged a new Add-in for Mindjet 11, we can build an installer for it.

The Add-in Wizard in Visual Studio helpfully creates a Setup project as part of the solution. We will not be explaining Visual Studio setup capabilities in detail here, but only pointing out some of the special considerations for installing

Mindjet 11 Add-ins.

The minimum requirements for installation of a Mindjet 11 Add-in are:

- Installing the correct files
- Configuring the registry to enable loading of the Add-in

Installing files

In the Solution Explorer in Visual Studio, browse the Setup project for your Add-in and go to Detected Dependencies. Right-click and refresh the dependency list. If you are not using VS2010, you should see Mindjet.MindManager.Interop.dll which is the API definition file. If this file is not excluded, right-click on it and exclude it, as it should not be included in the installation. The copy that is already registered in the GAC by Mindjet 11 should always be used. Visual Studio 2010 does not display the Interop file as a dependency and does not include it in the installer package by default.

Some versions of Mindjet 11 will also include MindManager.exe in the dependencies. If you see the executable in the list, right-click and exclude it. Assuming that your Add-in gets installed in its own folder, then including the executable there will not do any harm, but will unnecessarily inflate the size of the installer file.

Add-ins also require extensibility.dll to be installed in the same folder as the Add-in DLL itself. If this file is omitted, the Add-in will not load. Right-click on the Setup project in the Solution Explorer, and go to View > File System. Browse to the Application Folder, which should already contain the output from the DLL project in this solution. Right-click in the Application folder listing pane and add a new file. Browse to the MindManager installation folder on the development system, and include a copy of the file extensibility.dll from this folder. You can also find this file in the Visual Studio common files - all versions tend to be the same as it is a well-established interface.

If your Add-in requires other resources, such as graphics files for ribbon buttons, you can add them to the Application folder.

Configuring the Registry

The minimum requirements for Mindjet 11 to be aware of an installed Add-in is an entry under **HKLM/Software/Mindjet/MindManager/11/AddIns** where **11** is the major version. There should be a key which is the exact same name as the value of "ProgIdAttribute" defined when the Connect class is declared. If the name of the key does not match the name of this class, then the Add-in will fail to load. If you successfully configured the Visual Studio Add-in Wizard earlier in this tutorial, then you should find that the key and values have been automatically created in the Setup project, and you do not need to do anything else. If the Wizard was not used, or if you are manually building a setup project for a Mindjet 11 Add-in, then the installer should create the following values:

- Key: HKLM/Software/Mindjet/MindManager/11/AddIns/AddInName where "11" is the major version, and "AddInName" is as described above. This key should contain three values:
  - SZ string "FriendlyName" containing the name of the Add-in as it will appear in the listing in the Mindjet 11 Options dialog, in the Add-Ins page
  - SZ string "Description" containing the text description of the Add-in as it will appear in the Options dialog when the Add-in is selected
  - DWORD value "LoadBehavior", set to the value 2, which will cause Mindjet 11 to attempt to load the Add-in at startup.

Assembly Name (VS 2010)

If you are using Visual Studio 2010, check that the Assembly Name for the DLL project has been populated. If the Wizard has left the Assembly name as the default "Project1" then Mindjet 11 will fail to connect to it at startup. The Assembly name should correspond to the root of the ProgIDAttribute declared in the Connect module. For example, if this is "MyAddIn1.Connect", then the Assembly name should be "MyAddin1".

Launch Conditions

For Mindjet 11, check that the target .Net Framework for the DLL project is also called up in the prerequisites and checked in the .Net Framework launch condition. In all cases, this should be .Net Framework 4, and not .Net Framework 4 Client Profile. (The Mindjet 11 host targets .Net Framework 4, so there is no value in independently installed Add-ins targeting anything different.) If different versions of the .Net framework are referenced in the DLL project and the installer project, an error will be shown when building the installer.

Optional configuration items

There are some other configuration items that are not mandatory, but which may make life a little easier:

- In the Properties pane with the Setup project selected, set **RemovePreviousVersions** to True
- In the same Properties pane, set **InstallAllUsers** to True

Since Mindjet 11 Add-ins depend on specific versions of Mindjet 11 , it makes sense to automatically check whether the right version is installed on the target system before installing the Add-in. To test for the presence of Mindjet 11, you can search for a Mindjet 11-related registry value, the presence of which would show that Mindjet 11 has been installed and executed at least once. In our example, we will use the registry value that points out the User Library, which is only published on the first run of Mindjet 11. To configure this in your Setup project:

1. Right-click on the Setup project for your Add-in in the Solution Explorer
2. Choose View > Launch Conditions
3. Browse to "Requirements on Target Machine", right-click on "Search Target Machine" and choose "Add Registry Search"
4. Name the registry search "Search for MindManager Library" then right-click on it and go to the Properties Window
5. Configure the following properties for this registry search:
   a. Property = MMLIBRARY
   b. RegKey = Software\Mindjet\MindManager\11\Settings (where "11" is the major version of the installation you are checking for)
   c. Root = vsdrrHKCU
   d. Value = LocalLibraryDirectory
6. Now right-click on "Launch Conditions" and choose "Add Launch Condition"
7. Name this launch condition "MindManager Library present" then right-click on it and go to the Properties Window
8. Configure the following properties for this launch condition:
   a. Condition = MMLIBRARY (returns True if the above defined property MMLIBRARY is not empty)
   b. InstallURL = http://www.mindjet.com (or any appropriate URL for obtaining Mindjet 11 for Windows)
   c. Message = "A valid installation of Mindjet 11 for Windows was not found. Mindjet 11 must be installed and run at least once before this Add-in can be installed. Click Yes to visit the Mindjet web site, or click No to cancel." This message is displayed and the installation is aborted if the above Condition evaluates to False.

This combination of a registry search and a launch condition that depends on it will result in the message and URL being displayed to the user if the installer does not find evidence of the required version of Mindjet 11 being already installed on the user's system.

Ready to build

You can now right-click on the Setup project in the Solution Explorer and choose "Build". This will rebuild the DLL project if necessary, then build a setup file containing the Add-in. The .msi file can be distributed to Mindjet 11 for Windows users. You may want to add further features such as a license agreement, read-me-first information and so on. These are normal Visual Studio features that are not described here.

6.4 Troubleshooting

There are many ways that a Mindjet 11 Add-in can fail, and some are harder to diagnose than others. Below you will find some suggestions for the more common problems, grouped into four areas:

- Failure to load the Add-in
- Failure to connect to the Add-in
- Typical runtime failures
- Failure to disconnect from the Add-in or to close down

Failure to load the Add-in

"Loading" is the first step before Mindjet 11 tries to connect to the Add-in. The symptoms of failure are either that there is no evidence of the Add-in having been loaded, or that Mindjet 11 displays an error dialog stating that the Add-in could not be loaded and will be disabled. In the first case, you do not see an error message, but cannot find any sign of the Add-in such as an expected custom ribbon tab, command buttons or other features. Typical causes are:

- The installation of the Add-in has not been completed by the end-user
- The end-user has installed the Add-in but did not actually have the required version of Mindjet 11 on their system
- There has previously been a problem with the Add-in and Mindjet 11 has disabled it. Check that the Add-in is enabled in the Mindjet 11 Options dialog, in the Add-ins page. If it throws an error when it is re-enabled, then this is the root of the problem
- The LoadBehavior value for this Add-in in HKLM might be set to 3 instead of 2. Mindjet 11 will think that the Add-in is already loaded and will not reload it.
- The original installation of Mindjet 11 may be faulty, and it might be missing registration of some internal components. It is always worth trying a repair of the Mindjet 11 installation if all else fails.

If Mindjet 11 encounters an error when trying to load the Add-in, it will display a generic error message and disable the Add-in from future loading attempts. The error message is not specific and does not give any clue as to the cause of failure to load. Typical causes are:

- The file **extensibility.dll** has not been included in the Setup, and is not present in the same folder as the Add-in DLL file. The solution is to include the file extensibility.dll and rebuild the Setup project.
- The option to make the Add-in COM-Visible has not been enabled in the DLL project properties > Assembly tab > Assembly Information.
- The name of the Add-in declared under HKLM/Software/Mindjet/MindManager/X/AddIns does not precisely match that declared by the "ProgIdAttribute" in the Connect.vb module, meaning that Mindjet 11 cannot locate the DLL. A symptom of this is a lack of file version information in the Mindjet 11 Options dialog > Add-ins page, when your Add-in is selected.
- The MindManager Interop library might not be correctly registered in the GAC. This can be diagnosed (but not fixed) by allowing the setup to deploy the file Mindjet.MindManager.Interop.dll to the Application folder. This is not a recommended or shippable solution, though, as it will result in incorrect operation of Add-ins depending on their load order.
- The Add-in might have been compiled against a newer version of Mindjet 11 than the target version. For example, if you build an Add-in using service pack 2 of a version of Mindjet 11, and the user installs on service pack 1, then the Add-in may fail to load. For maximum compatibility in the field, use the oldest service pack version of Mindjet 11 that you can for building Add-ins, but of course also test it on the newer versions.

Failure to connect to the Add-in

Your Add-in may load successfully, but may fail to connect. Mindjet 11 will display an error message saying that it could not connect to the Add-in, and will unload it and disable it from further attempts to load. This is usually caused by an unhandled exception in the Connect.OnConnection() method. All exceptions should be handled so that they do not escalate to Mindjet 11.

Typical runtime failures

When the Add-in is loaded and running, the number of possibilities for error are limited only by your imagination.

Most of them make themselves known by unhandled exceptions being escalated to Mindjet 11, causing it to "crash" or terminate unexpectedly. Users are very unappreciative of this, as it often results in lost work. Mindjet 11 can raise exceptions in unexpected places, even though you think the code should work without any problem. Microsoft .Net 4 (which is required by Mindjet 11 onwards) is stricter about errors than .Net 3.5 was. Some of the sources include:

- Using an out-of-range index for a collection. If you iterate over a collection with a procedure that can change the collection (such as deleting an item), you must restart the iteration each time. Otherwise, the iteration index will go out of range and raise an exception.
- Removing an icon from a topic that does not contain it, or adding an icon to a topic that already contains it.
- Attempting an unsupported operation on an object, even though no syntax error was flagged at compile time, leading you to assume that it was a legal operation. There are some areas in the MindManager object model where similar objects use a common enumeration set, but not all the values of the enumeration apply to certain types of object. Mindjet 11 catches this at run time rather that at compile time, leading to an exception.
- Modifying the map during an event, while there is a user dialog open in Mindjet 11. It is not difficult to get into this position if you use the Idle event to service changes in a map.
- Initiating a Transaction while a user dialog is open.
- If Add-in events do not seem to be firing properly, check that MindManager.exe actually quits properly on shutdown and does not simply reopen the UI against a stale process when next started. See the notes below related to Mindjet 11 failing to close down.
- Unhandled exceptions inside an event handler may not be escalated to Mindjet 11, causing it to crash, but the thread for the event handler may never return, causing Mindjet 11 to lock instead. Always catch exceptions inside Event handlers. Just because you do not see any outward signs of an exception, it does not mean that one has not occurred.

Fortunately, you can avoid many of the ugly experiences for the end-user by *always* catching exceptions, even in places where they do not seem likely.

Failure to disconnect from the Add-in or to close down

Your Add-in might load and run successfully, but when closing down Mindjet 11, you get an error stating that MindManager could not disconnect from the Add-in. Alternatively, the Mindjet 11 UI may close, but the MindManager.exe process may continue to run even after Mindjet 11 has been closed down. A symptom of the latter is that Mindjet 11 opens very quickly when restarted, but behaves unpredictably and may have Add-in commands missing or inoperative. The only escape is to terminate the MindManager.exe process from the Windows process monitor. Typical root causes include:

- Unhandled exceptions in the Connect.OnDisconnection() method.
- Leaving a MindManager object pointer open. You should always dispose of all local MindManager object pointers when you have finished with them, or in the case of object pointers with global scope, they should be disposed of in the Connect.OnDisconnection() method. It is far easier to build this discipline into your code than to try to find an open pointer in the source code when this problem has been encountered.
- Accessing an object in the OnDisconnection() method that has just gone out of scope, e.g. trying to squeeze in one last access to a Document that is closing or has already been closed. Assume that Add-in functions run in their own thread, and things may be happening in parallel.
- Leaving something to be handled by garbage collection, which depends on an object that may already gone out of scope (such as the MindManager Application object). Check that the .Dispose methods of all classes do not rely on the presence of any object pointers which could have been disposed of earlier.

Lastly, always check that your Add-in can be cleanly disconnected and reconnected during a Mindjet 11 session by disabling and re-enabling it in the Mindjet 11 Options dialog. If errors occur during these steps that do not occur when starting and closing Mindjet 11, then look for unreleased pointers than have left objects in scope on disconnection.

6.5 Best Practices

The requirements for a Mindjet 11 Add-in will depend to a large extent on the target audience. If you are developing for in-house use only, then you will not need to implement many of the features that would be needed for an Add-in intended for commercial deployment, such as a polished installer. If the function performed by your Add-in is fairly simple, then performance optimization might not be important, but if it needs to perform large scale changes in real time, then more design work will be needed on the performance aspects. A few of the typical features and performance considerations are discussed below.

Object pointer management

As mentioned in other places in this documentation, it is vital to properly discard all Mindjet 11 object pointers when disconnecting from an Add-in, otherwise MindManager.exe may not close.

Design for performance

Much of the information about Mindjet 11 and the documents under edit is constructed by Mindjet 11 when requested through the API. For example, when you access the collection of all Topics via a Document.Range object, remember that Mindjet 11 has to construct this collection on demand, which can be slow. Topic Notes are also dynamically converted to the requested format. To maximize performance, consider caching objects and properties returned from the API, if they will be used more than once in the same scope. For example, if you need to use the percent-complete value from a topic task more than once in a routine, then read it once and store the value locally, rather than read it twice from the API. If you are iterating over a large number of topics, this can have a significant effect on performance.

When making updates to the active map, always use a Transaction object. This will suppress the updates to the user interface until the transaction is committed, which occurs automatically when execution completes, unless you manually invoke it earlier. If your Add-in is working on a large map and making multiple updates to topic, then using a transaction will significantly improve performance.

Providing Undo capability

By default, Mindjet 11 will create an undo entry for each individual edit of a map. An Add-in can typically make composite changes to a map as a single operation, such as writing to a Custom Attribute as well as updating a visible property of the topic. Not only will the presence of an individual undo step for each action be confusing for the user, but if they undo one or two steps in a sequence that are designed to work together as a group, then it may leave the map in an invalid condition. The solution is to use a Transaction object to group together the changes, which creates a single Undo step for all the updates made within the transaction.

However, there may be occasions where the Add-in not only makes one or more updates to the current map, but also changes internal data. When the user carries out an Undo operation, there is no notification back to the Add-in, so there is no opportunity to undo the changes to the internal data as well. In this case, it is better to suppress the Undo completely rather than risk a mismatch between internal data in the Add-in and data in the map. (Commands that change the command state of Mindjet 11 are not usually undoable, e.g. modifying the Mindjet 11 options cannot be undone but must be manually reversed.)

Suggested integration points

Long initialization times can slow down the launch of Mindjet 11 and frustrate users. You can offset this by deferring initialization until demanded by the first use of the Add-in. If you use this technique to complete installation, e.g. by deferring the unpacking and placing of images or templates until they are first needed, bear in mind that this will leave resources behind when uninstalling the Add-in, which is also not appreciated very much. Getting a balance between tolerable load times and the footprint of unmanaged resources can sometimes be a compromise rather than design perfection.

If you use a splash dialog to indicate loading of your Add-in or a purchase reminder, include an option to disable it.

So that users can clearly see whether your Add-in is installed or not, a custom Ribbon tab is usually a good idea, even if there are not many commands to put on it.

If your Add-in has options that can be configured, create a page for it in the Mindjet 11 Options dialog, so that users can configure it alongside other Mindjet 11 options.

Ensure that the Add-in name unambiguously identifies it in the Add-ins page in the Mindjet 11 Options dialog, and the installer & uninstaller are clearly identified in the Windows Control Panel.

If you need to use non-modal dialogues that remain in place while the user continues to edit the map, consider using a Task Pane for this purpose. A Task Pane can use a much wider range of Controls than the ribbon.

Although the Mindjet 11 API allows you to integrate features and functions into Mindjet 11, you cannot also extend the Mindjet 11 documentation, so you will need to provide separate documentation for your Add-in.

If you use keyboard shortcuts for the ribbon buttons, bear in mind that your Add-in may be installed on different language platforms, and the availability of keyboard shortcuts will vary by language and by Mindjet 11 version. It will also depend on what other Add-ins might be installed. There is no central registry of adopted shortcut keys published by Mindjet, so it will be a trial and error process to find suitable keys.

Lastly, consider including an "About" button that displays helpful support information such as the version number of the Add-in and other basic configuration settings. The easier it is for users to access support information, the more likely it is that they will use it when requesting support.

7 Appendices

7.1 About this Documentation

Documentation version

This is build 642 of the API documentation. The Object Model documentation reflects the Object Model version 11.2.1 as published at Mindjet 11 v11.2.185. This documentation describes the API for Mindjet 11 for Windows, and highlights changes and additions in the API between MindManager 2012 and Mindjet 11.

Known Documentation Issues

- The syntax for the **Item** property of collections does not show the index argument. Generally, the syntax should be Collection.Item(ByVal index as long) as ItemType. This is commented on in the notes in most places and will be corrected at a future version of this documentation.
- Some of the object diagrams unnecessarily show the members of parent objects. These diagrams are created automatically, and a property pointing back to a parent object is indistinguishable from one pointing at a child object. In terms of understanding the members of a class, it is not very helpful to see the other children of parent objects.

Where to find updates

For updated versions of this documentation, consult the Mindjet Developer Network on the Mindjet web site (www.mindjet.com)

How to report problems

For developer support, consult the Mindjet Developer Network on the Mindjet web site (www.mindjet.com)

Terms of use

Access to and use of this documentation is subject to the Terms of Use published in the Mindjet Developer Network.

Copyright Info