API v2.0 for Antidote
Windows – COM

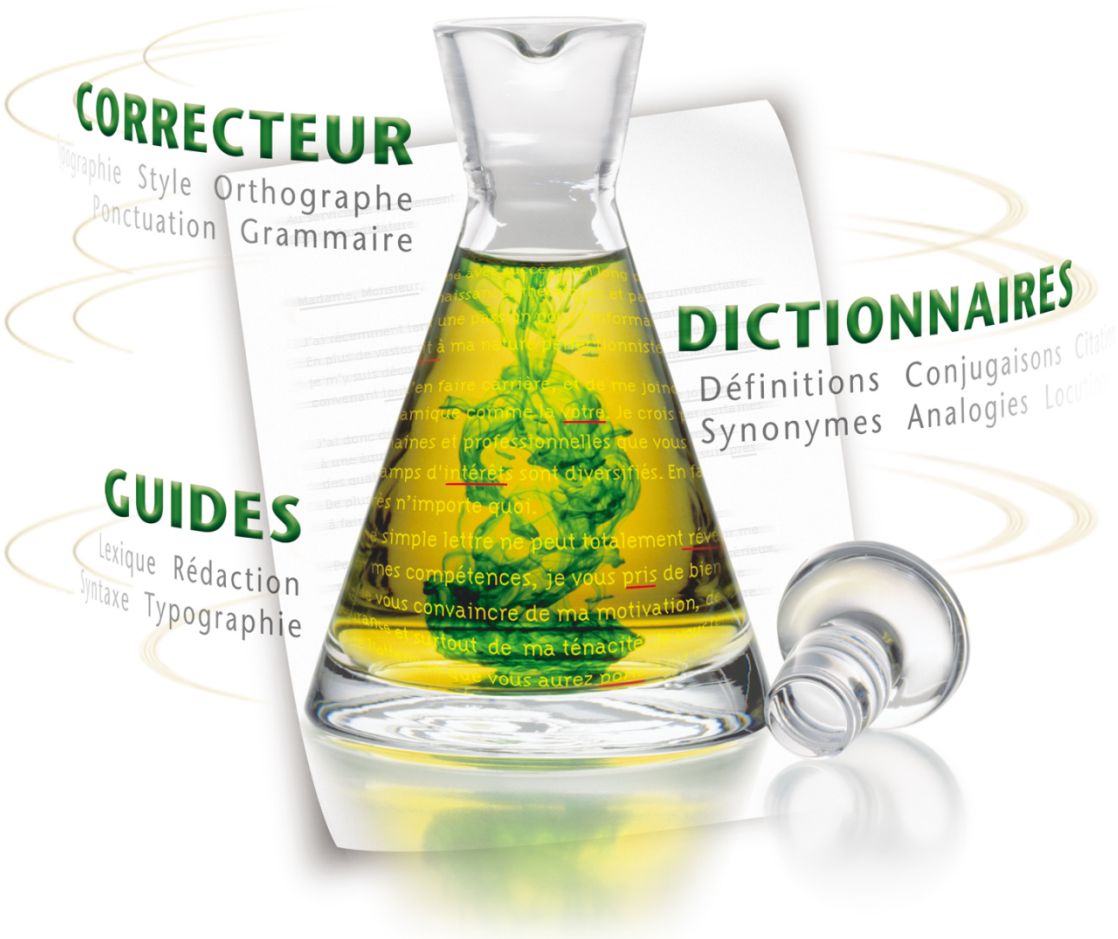

Table of Contents

Important

If your application already implements a version of Antidote's API, please see Section 8.

1. Introduction

Antidote's API is an interface that allows you to call Antidote's services from your Windows application. Our API is based on Microsoft's COM (Component Object Model) technology.

This document will explain how to establish communication between your application and Antidote.

2. Basics

a) Your application must allow access to text using character positions.
Example: Antidote may send a request to your application to return the text between characters 1023 and 1725.

b) Your programming infrastructure (language, libraries, etc.) must allow access Antidote's COM interface.

c) You need to be able to implement a COM interface in your application.

d) Antidote and your application must be running simultaneously. The API simply allows them to communicate with each other.

3. Estimating Time and Difficulty

Difficulty:
Easy if the basic concepts outlined above have already been applied.
Average if not.

Time:
Five person-days if the basic concepts outlined above have already been applied.
If not, it will depend on the complexity of the application.

4. Outline of the COM Interfaces

4.1 Accessing Antidote's COM Interface

To call Antidote's tools, you will need to use **the IApiOle interface** that Antidote exports. This is Antidote's only public interface. The other interfaces are for Druide's internal use and can change without notice.

The **IApiOle interface** contains the following methods. These methods are described in detail in Section 5.2.1. The other API methods, which are not listed here, exist solely for reasons of backward compatibility .

void **LanceOutil2**(BSTR classe, BSTR outil, BSTR langue, BSTR versionAPI);
void **LanceOutilDispatch2**(IDispatch* disp, BSTR outil, BSTR langue, BSTR versionAPI);

4.2 Creating the COM Interface in your Application

In order for Antidote to be able to communicate with your application, you will need to create a COM interface. This interface must be specific to Antidote.

The interface that you create will need to contain the following methods. These methods are described in detail in Section 5.3.2.

```
HRESULT ActiveApplication(void);
HRESULT ActiveDocument([in] LONG idDoc);
HRESULT DonneDebutSelection([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
HRESULT DonneFinSelection([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
HRESULT DonneIdDocumentCourant([out, retval] LONG *retour);
HRESULT DonneIdZoneDeTexte([in] LONG idDoc, [in] LONG indice, [out, retval] LONG *retour);
HRESULT DonneIdZoneDeTexteCourante([in] LONG idDoc, [out, retval] LONG *retour);
HRESULT DonneIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin, [out, retval] BSTR *retour);
HRESULT DonneLongueurZoneDeTexte([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
HRESULT DonneNbZonesDeTexte([in] LONG idDoc, [out, retval] LONG *retour);
HRESULT DonnePolice([in] LONG idDoc, [in] LONG idZone, [out, retval] BSTR *retour);
HRESULT DonneTitreDocCourant([out, retval] BSTR* retour);
HRESULT RemplaceIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin, [in] BSTR laChaine);
HRESULT SelectionneIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin);
```

5. Communication

5.1 Step One: Launch Antidote

Before you call an Antidote tool from your application, you will need to make sure that Antidote is open. If this is not the case, your application needs to launch Antidote.

Antidote can be found in the Windows registry in the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\Druide informatique inc.\Antidote

You will need to access the following value:

DossierAntidote

Your application should access this value to find Antidote's location and to run it with the following parameter: "-activex". Your application then needs to wait for Antidote to be completely open before continuing. To see if Antidote is open, your application can check for the existence of the Antidote window.

Example: Access the LancerAntidote() method in the AntidoteApiOle.h folder to see the implementation details (see Section 7).

A bridge version will need to be provided to Antidote. This version number will allow you to interact with different editions of Antidote (i.e. different versions of the API).
You will need to indicate the most recent API that was supported by both Antidote and your application. The version supported by Antidote can be found in the registry in the same key mentioned above. You will need to access the following value:

VersionAPI

For example,

if you implement API 2.0 and 2.1 in your software
and Antidote implements API 2.2,
you will then have to instantiate API 2.1 in your application and use 2.1 in LanceOutil.

5.2 Step Two: Call an Antidote Tool

Your application will use the methods available in the **IApiOle** interface to request the opening of an Antidote tool.

The **IApiOle** interface's progID is Antidote.ApiOle

Important: You need to find Antidote's COM server in the Running object table (ROT) rather than using ::CoCreateInstance(). The TrouverServeurCOMDansLaROT() method in the AntidoteApiOle.h folder demonstrates how to do this (see section 7).

5.2.1 List of Methods for Antidote's IApiOle Interface

void **LanceOutil2** (BSTR* szClientProgId, BSTR* szlOutil, BSTR* langue, BSTR* versionAPI);

Description: Sends a request to open a tool (corrector, dictionaries or guides).

Parameters: szClientProgId: the ProgID of the interface that you created for Antidote (See Section 4.2). Your COM server needs to be a singleton for this method, since Antidote will use ::CoCreateInstance() on this ProgID.
szlOutil: the name of the desired tool. Go to the constants in the namespace CstApiAntidote of the AntidoteApiOle.h folder to see the possible values.
langue: the language in which you wish to carry out your search. Go to the constants in the namespace CstApiAntidote in the AntidoteApiOle.h folder to see the possible values.
versionAPI: the version of the API that your word processor implements (2.0 in this case).

void **LanceOutilDispatch2** (IDispatch* p_dispatch, BSTR* szlOutil, BSTR* langue, BSTR* versionAPI);

Description: This method is similar to LanceOutil but it allows Antidote to use the IDispatch pointer so that it can call back your server.

Example: Consult the AntidoteApiOle LancerAntidote() class in the AntidoteApiOle.h folder to see the implementation details. (See Section 7)

5.2.2 Example in Pseudocode for Calling the Corrector

(See Section 8 for examples in C++ and in C#)

```
If (AntidoteHasLaunched() OR LaunchAntidote())
{
WaitUntilAntidoteIsReady();

AntidoteApiServer ant;
if(FindCOMServerInROT(ant)) // Running Object Table
{
ant.LaunchDispatchTool(
pointerOnOwnComServer,
CstApiAntidote::kModuleCorrecteur, // constant for the correction module
languageForThisSearch,
APIVersion,

          );
  }
}
```

5.3 Step Three: Answers to Antidote's Requests

In order to open a tool, Antidote needs to send certain requests to your application. Antidote will call the methods found in the COM interface that you created for Antidote in your application (See Section 4.2).

5.3.1 Identifiers for Documents and Text Boxes

If your document is divided into several text boxes (e.g. header, footnotes, etc.), Antidote can correct them at the same time as the rest of your document. Your application will need to manage a list of text boxes for every document. Antidote will call the DonneIdZoneDeTexte(LONG idDocument, LONG indiceZoneDeTexte) method for each of the document's text boxes by using the text box's index as a second parameter. The index starts at 1 and corresponds to the rank of the text box in your list. You will need to provide a unique identifier for the document as well as for each text box. Antidote will use these identifiers to send requests to a specific text box in a specific document.

If your document only contains one text box, you will need to provide separate identifiers for the document and the text box.

5.3.2 List of Methods Called by Antidote

HRESULT ActiveApplication(void);
Description: Activates your application to place it in the foreground

HRESULT ActiveDocument([in] LONG idDoc);
Parameter: idDoc: Document identifier.
Description: Activates the specified document to place it in front of the other documents of your application without placing your application in the foreground.

HRESULT DonneDebutSelection([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
Description: Returns the starting position of the selection for the **idZone** text box in the **idDoc** document. The position is calculated by character starting from the beginning of the text box.

HRESULT DonneFinSelection([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
Description: Returns the end position of the selection for the **idZone** text box in the **idDoc** document. The position is calculated by character starting from the beginning of the text box.

HRESULT DonneIdDocumentCourant([out, retval] LONG *retour);
Description: Returns the identifier of the current document (the document in the foreground of your application).

HRESULT DonneIdZoneDeTexte([in] LONG idDoc, [in] LONG indice, [out, retval] LONG *retour);
Parameters: idDoc: Document identifier.
Indice: Index for the text box in the document. The index begin at 1.

Description: Returns the text box identifier corresponding to the **indice** index in the **idDoc** document.

HRESULT DonneIdZoneDeTexteCourante([in] LONG idDoc, [out, retval] LONG *retour);
Parameter: idDoc: Document identifier.
Description: Returns the identifier of the current text box (the text box that contains the cursor).

HRESULT DonneIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin, [out, retval] BSTR *retour);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
leDebut: The position of the beginning of the interval compared to the beginning of the text box (in characters).
laFin: The position of the end of the interval compared to the beginning of the text box (in characters).
Description: Returns the text of the **idZone** text box of the **leDebut** position up to the **laFin** position in the **idDoc** document.

HRESULT DonneLongueurZoneDeTexte([in] LONG idDoc, [in] LONG idZone, [out, retval] LONG *retour);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
Description: Returns the entire length of the **idZone** text in the **idDoc** document.

HRESULT DonneNbZonesDeTexte([in] LONG idDoc, [out, retval] LONG *retour);
Parameter: idDoc: Document identifier.
Description: Returns the number of text boxes in the specified document.

HRESULT DonnePolice([in] LONG idDoc, [in] LONG idZone, [out, retval] BSTR *retour);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
Description: Obsolete method present only for reasons of backward compatibility. Returns an empty string.

HRESULT DonneTitreDocCourant([out, retval] BSTR* retour);
Description: Returns the title of the current document.

HRESULT RemplaceIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin, [in] BSTR laChaine);
Parameters: idDoc: Document identifier.
idZone: Text box identifier.
leDebut: The position of the beginning of the interval compared to the beginning of the text box (in characters).
laFin: The position of the end of the interval compared to the beginning of the text box (in characters).
laChaine: The string of replacement characters.
Description: Replaces the text in the **idZone** text box in the **leDebut** position up to the **laFin** position with **laChaine** in the **idDoc** document.

HRESULT SelectionneIntervalle([in] LONG idDoc, [in] LONG idZone, [in] LONG leDebut, [in] LONG laFin);
Parameters: idDoc: Document identifier.

idZone: Text box identifier.
leDebut: The position of the beginning of the interval in relation to the beginning of the text box (in characters).
laFin: The position of the end of the interval compared to the beginning of the text box (in characters).

Description: Selects the text of the **idZone** text box of the **leDebut** position up to the **laFin** position in the **idDoc** document (to show Antidote's progress).

6. Activating the Log

To make it easier to develop your COM server, you can activate a log that will contain records of all the calls between Antidote and your application. The journal will also contain any exceptions (COM errors), if need be.

Note: The log will significantly slow down communication between the two applications and should only be used when you are developing your COM server.

To activate the log:

1. Open the Config.xml file with a text editor that can handle Linux carriage returns (Wordpad, for example).
   The default path for this file is the following:
   C:\Programmes\Druide\Connectix 10\Application\Config
2. Add the following entry into the <Choix> section.
   Please note that the entries in this file are case-sensitive.
   ```
   <Choix>
   <JournalApiCOM>1</JournalApiCOM>
   </Choix>
   ```
3. Quit and then relaunch AgentConnectix.
4. The log will be created during the next interaction with your COM server. Its location is:
   C:\Users\[user name]\AppData\Roaming\Druide\Connectix\JournalApiCOM.txt

7. Complete Example (C++ and C#)

The compressed file containing this PDF also contains two examples of a generic text editor which can be integrated with Antidote. One is in C++ and the other is in C# (source code and executable). These examples are a good starting point for integrating Antidote into your application. You can copy and paste a large portion of the example directly into your application.

For more details, please see the **"Readme.txt"** document in the examples folder.

8. API Compatibilities

Since Antidote's COM API's are backwards compatible, you can write your application so that it is compatible with multiple editions of Antidote by using the API that corresponds to the oldest edition.

The first COM API was available in Antidote Prisme (5).
The DDE API provided with Antidote MP (4) is no longer supported.

New features of Antidote 9's API (API 2.0) compared with Antidote HD (7) and Antidote 8's API. :

- The “bridge version” is provided so that you can target multiple versions of the API.
- You can make a request to open the dictionaries and guides for a word in a specific language.

New features compared to AntidoteRX's (6) API:

- Antidote must now be launched with the parameter « -activex ».

New features compared to Antidote Prisme's API

- There is now only one method LanceOutil instead of five (corrector, dictionary, synonyms, conjugation and grammar).
- You must now connect to Antidote using "Running object table" (ROT) instead of ::CoCreateInstance().
- It is recommended that you use the call method variants for Antidote with Dispatch.

