ASTRA Automation API
Getting Started

Contents

1 Introduction

Whenever tasks are repetitive or require minimal human intervention, the ASTRA Automation API enables automation to alleviate human effort. In addition, you are in full control of your data, and you can store experiment data files, data traces, results into your preferred format.

This integration can automate the following:

- Create new experiments from predefined ASTRA methods.
- Open/save experiment files.
- Collect data via individual measurements.
- Set collection and analysis parameters (e.g., baselines and peaks).
- Extract data and results from experiments.

Note: The ASTRA Automation API interface does not cover all the ASTRA functionality. It is meant for automating the collection of data and performing basic operations to extract data and results.

To perform the automation using the ASTRA Automation API, it is necessary to have a basic programming knowledge. It works with all programming languages supporting the Microsoft™ COM technology, nevertheless we recommend any programming language targeting the Microsoft™ .NET platform or python. In this document and elsewhere, the COM interface references the underlying implementation for the ASTRA Automation API.

2 Activation Code

Before you can start using the ASTRA automation API, you need to enter an activation code in the feature activation dialog. To do this, start ASTRA. If no activation code is present, the dialog will appear upon first launch, otherwise activate it via the *System/Feature Activation …* menu entry.

When you enter the activation code into the feature activation dialog, there will be a green checkmark in the *Activation* column next to the *ASTRA Automation API* feature if your activation code has this license. If your activation code lacks this license, you will have a red X mark in the *Activation* column and all calls to the COM interface methods will fail.

Do this on any computer where you will be using the COM interface.

3 Security Pack

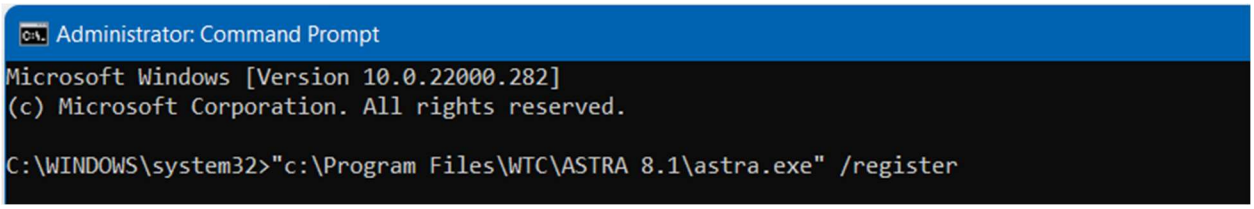
The ASTRA Automation API supports ASTRA in Security Pack mode and requires a user to log on before performing any operations involving opening or saving an experiment. See Security Pack Logon on how to perform the logon operation.

4 COM Interface Registration

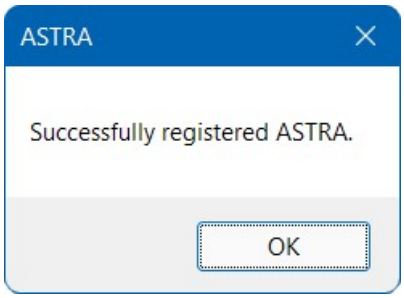
Normally, ASTRA registers the COM interface upon installation. If you have multiple versions of ASTRA installed, only the most recently installed version of ASTRA is active (note that it could be an older version of ASTRA, e.g., installing 8.2 and then installing 7.3).

If you are getting a *"Class Not Registered"* error, then you should manually register the COM interface.

To do this, you need to run a registration command from an elevated command prompt with the argument */register*:

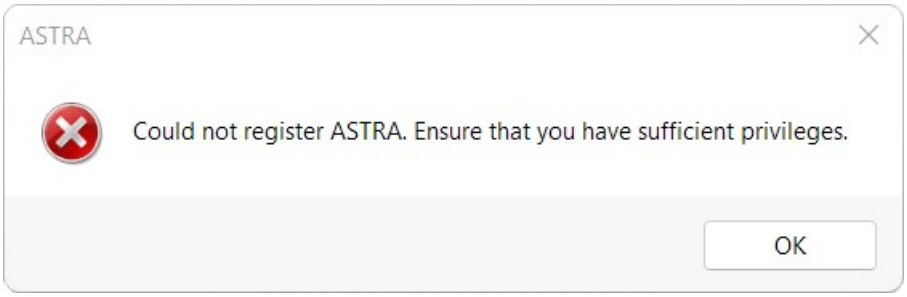


After pressing enter, ASTRA will register itself and display a confirmation dialog such as:



At this point the COM interface should be registered.

If you do not run this command in an elevated command prompt, you will get an error such as this:



5 Platform Setup

Since the ASTRA Automation API uses a COM interface, it can be used from a variety of platforms. The only requirement is that the platform runs on Windows and has support for creating COM clients. There are details for using the COM interface from .NET below. For other platforms and programing languages, you will need to know how to create COM clients either from a registered COM interface or an IDL file.

5.1 .NET

A .NET/COM interop assembly is provided with the installation package. From here on, we assume that the reader has enough knowledge of .NET to add an assembly reference to a project file and to create objects, call methods, and subscribe to events in their language of choice (sample code uses C#).

To use the interop assembly, just reference the *AstraLib.dll* assembly in your .NET project. Then create an `Astra` class instance as shown just below. This will cause ASTRA to start if it isn't already running.

```
using AstraLib;

public class Main
{
    public Main ()
    {
        Astra = new Astra ();
    }

    Astra Astra;
}
```

The object will have methods for all methods in the COM interface and *Event* members for all the events in the COM interface. The `AstraLib` namespace also contains the other types defined in the COM interface.

5.2 Python

In addition to a native .NET implementation, we are also providing a python implementation. Although python supports COM, it is tedious, and we are including a python library to easily use the ASTRA Automation API. It is located under the SDKPython, requires Python 3.11 or later, and the following components to be installed:

```
python -m pip install comtypes
python -m pip install setuptools
python -m pip install psutil
```

Once the prerequisite is installed, you can write your own python program by importing the astra_admin module:

```
from astra_admin import AstraAdmin
```

The `AstraAdmin` object offers the same functionality as the code available for .NET (see AstraAdmin).

6 API usage

6.1 Starting ASTRA

Only one instance of ASTRA can be running at a time, so before starting ASTRA via the COM interface, you need to check that ASTRA is not already running. If it is running, you should request it to be closed. To see if it is running, a mechanism has been set to allow clients of the COM component to check if it has either been started normally (i.e., a user starting it) or via COM.

Call `IsEmbedded` and if it returns 0 then ASTRA was started normally. If it returns 1, you need to check if the COM instance is new or if there is already a COM client. To do so, call `GetAutomationUid`, and it will return a Guid. If the Guid instance is the empty Guid, you are the only COM client, otherwise another COM client is already controlling ASTRA. This other COM client needs to be closed before you can proceed.

6.2 Controlling the Instrument

The COM interface has methods for controlling the instrument state. This allows you to set up the instrument state before each measurement if you are doing a single measurement collection. These commands should not be used while an event schedule or experiment designer method is running. As an

example, there is the `SetPumpFlowRate` method which takes an experiment ID and the pump flow rate to set the instrument. There is an associated `GetPumpFlowRate` command that takes an experiment ID and returns the current pump flow rate. Most instrument control commands follow this pattern. Please see the COM interface reference document for details.

6.3 Getting and Setting Baselines/Peak Ranges

The following functions `GetBaselines`, `UpdateBaselines`, `GetPeakRanges`, `AddPeakRange`, `UpdatePeakRange`, `RemovePeakRange` can be used to get/set various properties on the experiment baselines and peaks.

6.4 Extracting Data and Results

Once an experiment has data and results (results are calculated automatically), methods can be used to extract information from the experiment.

The `GetDataSet`, `SaveDataSet` methods can be used to retrieve data set information for a given experiment. An experiment ID and data set definition name is needed to retrieve the information. The data set information is in a CSV-like format.

The `GetResults`, `SaveResults` methods can be used to retrieve results for a given experiment. An experiment ID is needed to retrieve the information. The results are in XML format.

6.5 Collecting Data

The COM interface allows you to collect data as a single experiment.

6.5.1 Creating an experiment from a method

Before any data can be collected, an experiment needs to be created. This can be done by calling `NewExperimentFromTemplate` by providing the path to an existing ASTRA method. This assumes that the method has already been created in ASTRA prior to this call. Upon success, it will return an experiment ID (an opaque identifier for the experiment).

Note: The experiment ID is core to manipulating experiment in the programming interface. Most of the exposed functionality of ASTRA will always require an experiment ID. If the wrong experiment ID is provided, the call will fail.

6.5.2 Starting the collection

Once an experiment has been created from a method, and prior to starting the collection, you can set/override some of the method's parameters such as injected volume, pump flow rate, duration, etc…

To start the collection, call the `StartCollection` method with the experiment ID you previously obtained.

6.5.3 Saving the Experiment

To save the collected data, you will need to save the experiment. This is accomplished using the `SaveExperiment` method. Pass it the experiment ID and the path to save the file to. Once you are done with an experiment, it can be closed via the `CloseExperiment` method where you pass it the experiment ID.

6.6 Event handling

Most operations in ASTRA are asynchronous and the proper way to know they completed is to use an event notification system. Examples of asynchronous operations are loading/saving an ASTRA data file, starting a collection, …

The following event handlers have been defined:

- `ExperimentClosed`
- `ExperimentRead`
- `ExperimentWrite`
- `ExperimentRun`
- `PreparingForCollection`
- `WaitingForAutoInject`
- `CollectionStarted`
- `CollectionAborted`
- `CollectionFinished`
- `InstrumentDetectionCompleted`

Except `InstrumentDetectionCompleted`, all events take the experiment ID of the experiment that triggered the notification.

It is required to wait for the `InstrumentDetectionCompleted` event before starting any collections and it is recommended to wait for this event after creating the `Astra` class instance.

6.7 Closing ASTRA

There is no command to shut down ASTRA, as it is automatically handled by the COM runtime. Nevertheless, when you are done using ASTRA, you need to issue a call to `RequestQuit`. If ASTRA was started in response to your client, then when your client exits, ASTRA will exit.

7 Using AstraAdmin in C#

7.1 Introduction

The ASTRA Automation API package comes with a small abstraction layer `SDKCommon.AstraAdmin`. This closely maps to the underlying ASTRA API but provide helper functions to make some complex operations much easier to perform. This abstraction layer is used in the samples that come with the ASTRA Automation API package.

7.2 Basic usage

The `AstraAdmin` class is meant to be used as a singleton. It is thread-safe (by using synchronization objects), however, it is recommended to use the `AstraAdmin` instance in a single thread. Using it in multiple threads would made the event callbacks triggered by the underlying API harder to manage. For the same reason, it is recommended to manipulate only one experiment at a time.

7.3 Initialization

The proper way to get started with `AstraAdmin` is to access the singleton via the `AstraAdmin.Get` method:

```
using SDKCommon;
using System;
using System.Diagnostics;

…

// Initialize the Astra instance by providing the name of the ASTRA
// client, its version, its process ID and a unique GUID identifying the
// client.
var process = Process.GetCurrentProcess ();
AstraAdmin.Get.SetAutomationIdentity ("Automation Client",
    "1.0.0.0", process.Id, $"{Guid.NewGuid ()}", 1);
```

Once ASTRA starts, you must wait for all the instruments to be loaded before doing any collection. You do this by calling `WaitForInstruments`:

```
AstraAdmin.Get.WaitForInstruments ();
```

7.4 Security Pack Logon

If Security Pack has been enabled in ASTRA, you need to logon (as a user would do when using ASTRA directly) before using the ASTRA Automation API. This should be done after the initialization step above using `AstraAdmin` and the `ValidateLogon` method:

```
var result = AstraAdmin.Get.ValidateLogon (“username”, “password”, “domain”);

// result.isValid return 0 if logon failed, return 1 if succeeded.
if (result.isValid == 0)
{
    // Process logon error.
}
```

It is up to the COM client to check the result of the call to `ValidateLogon`. If the logon failed, the returned value also contains an error message with an explanation for the logon failure. We recommend prompting again for the user’s credential and to not continue the execution of the code until it has been validated. It is up to the COM client to decide how many times they should try the logon validation.

7.5 Collecting Data

A convenience method `CollectData` is provided in `AstraAdmin` to take care of the collection from start to finish. It requires the method to use to collect data, the path where to save the ASTRA experiment files (without the ASTRA extension) and various other parameters such as the sample, the duration of the collection, the injection volume, the flow rate and a callback function to report progress.

It can be used to easily create your own sequences.

7.6 Event Handling

Having to wait for events (See Event Handling) can be cumbersome when using event handlers. To make it easier, the following helper functions have been added:

- WaitForInstruments
- WaitExperimentClosed
- WaitExperimentRead
- WaitExperimentRun

- WaitExperimentWrite
- WaitWaitingForAutoInject
- WaitCollectionFinished
- WaitCollectionStarted
- WaitPreparingForCollection

8 Using AstraAdmin in python

8.1 Reading documentation

To avoid too much documentation duplication, all the ASTRA Automation API uses C# as the example programming language. Adapting the various calls to python is mostly a matter of style convention. Whereas C# uses CamelCase for member functions, python will use snake_case. For example, `SetAutomationIdentity` becomes `set_automation_identity`, `WaitForInstruments` becomes `wait_for_instruments`.

To make it more obvious of the style difference, we are translating below paragraph 7.3 for C# into python. For other calls, please refer to the C# code's documentation.

8.2 Initialization

The proper way to get started with `AstraAdmin` is to access the singleton via the `AstraAdmin` method:

```
import os
import uuid

from astra_admin import AstraAdmin

client_id = uuid.uuid4()

AstraAdmin().set_automation_identity(
       "SDK Command Line App", "1.0.0.0", os.getpid(), client_id, 1)
```

Once ASTRA starts, you must wait for all instruments to be loaded before doing any collection. You do this by calling wait_for_instruments:

```
AstraAdmin().wait_for_instruments()
```