

VisualCron API

VisualCron API	1
Purpose.....	2
COM support	2
VB6 example.....	2
VB6 code sample	2
Architecture.....	2
Object model	3
Methods	3
Events	4
Communication	5
Local.....	5
Remote	5
Requirements	5
License	6
Getting started	6
Samples	6
Documentation.....	6
Support & Questions	6

Purpose

Allow local or remote applications to access the functionality in the VisualCron server through an easy-to-use interface.

COM support

The API is not only for .NET usage. You can use any other language that supports COM. Here are the steps you need to perform besides the normal Requirements. First, download REGASM tool:

[http://msdn.microsoft.com/en-us/library/tzat5yw6\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/tzat5yw6(VS.80).aspx)

VB6 example

1. Copy VisualCron.dll and VisualCronAPI.dll to folder where you have your vb.exe (if you are debugging use C:\Program Files\Microsoft Visual Studio\VB98 (where vb6.exe resides)).
2. Run regasm.exe on VisualCron.dll and VisualCronAPI.dll like this: regasm VisualCron.dll /tlb:VisualCron.tlb
3. Add reference in Project->References to both *.tlb files
4. Write your code
5. Run or build output file
6. When moving your output file you need to include the two *.dll files
7. You can safely remove *.dll and *.tlb files from C:\Program Files\Microsoft Visual Studio\VB98 folder
8. In order to upgrade to a new version you need to perform step 1 and 2

VB6 code sample

```
Private Sub Form_Load()  
    Dim c As New VisualCronAPI_Client  
    Dim s As VisualCronAPI_Server  
  
    Dim conn As New Connection  
    conn.ConnectionType = ConnectionT_Local  
    Set s = c.Connect(conn, True)  
    MsgBox s.InSync  
End Sub
```

Architecture

There are 3 important files when you are working with the API:

- VisualCron.dll – this file contain all shared class objects that a Client and Server should know about. For example, you have the base class JobClass which contains all member of a JobClass. Detailed information of member are available in the help file VisualCron.chm

- VisualCron_nat.dll – this file is a protector file to VisualCron.dll and should just remain in the same directory as VisualCron.dll
- VisualCronAPI.dll – this file is core for the API. You reference to this file and get access to all functions regarding Connecting to a Server and modify objects on the Server.

Object model

As you can see in the sample(s) you use a Client-object to Connect to a VisualCron Server. What happens is that, upon successful connection, all Server objects are transferred to the Client. When that “sync”-procedure is done the Client will return a Server-object. The Server-object is a direct access object to all the Server functions. You can Add Jobs, Tasks and Update all kind of information in the Server that you are connected to. Here is the current Server model:



During the connection you will receive updates from the Server. When a Job changes you will get that change and your Jobs in the Jobs object will be updated.

Methods

All objects (Jobs,Conditions,Connections etc.) share almost the same methods:

- Get – return a specific object based on Id
- GetAll - returns all objects
- Add – adds a new object
- Update – updates a new object
- Remove – removes an object

- RemoveAll – removes all objects

Basically, you access those from the Server object like this (which you can see in the samples):

```
Server.Jobs.Add(JobClass)
```

Jobs Class
Namespaces ► VisualCronAPI ► Jobs

C#



All Members	Constructors	Methods	Properties	Events
<input checked="" type="checkbox"/> Public		<input checked="" type="checkbox"/> Instance		<input checked="" type="checkbox"/> Declared
<input checked="" type="checkbox"/> Protected		<input checked="" type="checkbox"/> Static		<input checked="" type="checkbox"/> Inherited
Icon	Member	Description		
≡	Activate(JobClass)	Activates a Job		
≡	Activate(String)	Activates a job		
≡	Add(JobClass)	Adds a Job		
≡	Contains(String)	Returns true if Job id exist		
≡	DeActivate(JobClass)	Deactivates a Job		
≡	DeActivate(String)	Deactivates a Job		
≡	DoJobNameExist(String)	Checks if name of Job already exists		
≡	Equals(Object)	Determines whether the specified Object is equal to the		

Events

To know exactly when a Job updates you can add one or more of the public events. There are public events to all kind of objects in the Server which have almost the same features:

```
public class Jobs : Jobs._Jobs
```

Members

All Members	Constructors	Methods	Properties	Events
<input checked="" type="checkbox"/> Public		<input checked="" type="checkbox"/> Instance		<input checked="" type="checkbox"/> Declared
<input checked="" type="checkbox"/> Protected		<input checked="" type="checkbox"/> Static		<input checked="" type="checkbox"/> Inherited
Icon	Member	Description		
	EventJobRemovedStatic	This event is raised when a Job has been removed		
	JobAdded	This event is raised when a Job has been added		
	JobAddedStatic	This event is raised when a Job has been added		
	JobRemoved	This event is raised when a Job has been removed		
	JobUpdated	This event is raised when a Job has been updated		
	JobUpdatedStatic	This event is raised when a Job has been updated		

There are both static and non-static events. For example, if your Client is connected to many Servers you may want to use the static event to get all Server events.

Communication

Different connection types exist where "local" is the primary.

Local

Local uses a connection type IPC. The server "listens" to a predefined pipe. The "client" creates a dynamic pipe from its id. The server is always fixed the client is dynamic so many clients can exist on one machine. At connect time it gives the server its own pipe address so that the server can answer back. The server keeps track of all clients from a dictionary. If it fails to send a message to the client the client is disconnected and removed from the dictionary. The local Connection is about 10x faster than the remote connection that uses sockets for the underlying communication.

Remote

The remote connection uses secure sockets for communication. By default, the server is listening to port 16444. This may previously have been changed by you. You can see which port to use (and change it) by opening the server settings. You need to specify Address property of Connection if you want to connect remotely.

Requirements

- .NET Framework 2.0
- The files in the API folder of the VisualCron install

License

You may use the API for free. To connect to the VisualCron Server you need a valid license installed – on the Server.

Getting started

Samples

Samples of how to connect, retrieve and change certain information exist in the samples folder.

Documentation

The documentation for the API is divided into several parts:

- This document
- A help file containing API reference to VisualCron classes (VisualCron.dll) and VisualCron API (VisualCronAPI.dll)

Support & Questions

Please use the forum for further dialog and if you have questions, found bugs, have enhancement requests etc.