

# Aufgabenplanung API

## Inhaltsverzeichnis

---

Einführung .....	3
Erste Schritte .....	3
Funktionsweise der API .....	3
Klassen, Funktionen & Eigenschaften .....	3
Verbindung zum Server aufbauen .....	5
Aufgabe(n) abfragen .....	6
Neue Aufgabe anlegen/bearbeiten .....	6
Klassenansicht .....	7
Information zur API .....	8

## Einführung

---

### Erste Schritte

Die API bietet Ihnen die Möglichkeit auch von Ihren Programmen schnell und effizient Aufgaben anzulegen und diese zu verwalten.

Damit Sie die API in Ihr Projekt einbinden können, müssen folgende Dinge zutreffen:

1. Ihr Projekt muss mindestens .NET Framework 4.5.2 einsetzen
2. Ihr Projekt muss eine Verbindung zum AufgabenServer herstellen können
3. Der AufgabenServer muss mit der selben Protokollversion arbeiten, wie die API

Eine Verbindung zum AufgabenServer ist zwingend notwendig, da dieser aufgrund von Firebird Embedded der einzige Prozess ist, der auf die Datenbank zugreifen kann.

### Einbinden in Ihr Projekt

Klicken Sie in Visual Studio im Projektmappen Explorer mit der rechten Maustaste auf *Verweise->Verweis hinzufügen->Durchsuchen* und wählen Sie die Datei AufgabenLibrary.dll aus. Diese befindet sich standardmäßig im Installationspfad des Servers, sowie der Aufgabenplanung.

Anschließend müssen Sie folgende Namespaces einbinden:

```
using AufgabenLibrary;
using AufgabenLibrary.Library;
```

Anschließend muss eine Objekt der Klasse "AufgabenApi" erzeugt werden :

```
AufgabenApi aufgabenApi = new AufgabenApi();
```

Die API ist nun eingebunden und kann nun verwendet werden.

## Funktionsweise der API

---

### Klassen, Funktionen & Eigenschaften

Diese Liste soll keine vollständige Liste aller Klassen und Funktionen wieder spiegeln. Sie soll einen Überblick über Grundfunktionen der API bieten.

Alle Klassen und Funktionen sind im Quellcode ausreichend Dokumentiert und können mithilfe von Intellisense einfach verstanden werden.

#### Klassen:

Klassenname	Zweck
<i>AufgabenLibrary.Library</i> <b>.Aufgabe</b>	Erstellt eine Leere Aufgabe mit Leerwerten, alle nötigen Felder haben somit einen Wert, damit es beim Eintragen der Aufgabe nicht zu abstürzen kommen würde.
<i>AufgabenLibrary.AufgabenApi</i> <b>.Arbeitsgruppen</b>	Ruft ein Array mit allen Arbeitsgruppen ab.
<i>AufgabenLibrary.AufgabenApi</i> <b>.Mitarbeiterliste</b>	Ruft eine Array mit allen Mitarbeitern, die den Status aktiv haben.
<i>AufgabenLibrary.AufgabenApi</i> <b>.DieserMitarbeiter</b>	Ruft alle Daten über den eingeloggten Mitarbeiter ab.

<i>AufgabenLibrary.GlobaleVariablen.Mitarbeiter</i>	Ruft ein Dictionary der Mitarbeiter ab - <UserID, Username>
---	---

*kursiv geschriebene Variablen sind nicht standardmäßig Initialisiert. Diese werden erst nach einer Verbindung zum AufgabenServer mit Daten befüllt.*

## Funktionen:

<b>Funktionname</b>	<b>Parameter</b>	<b>Zweck</b>
AufgabenLibrary.AufgabenApi. <b>AufgabenApi()</b>	<i>string</i> <b>ServerIp</b> , <i>int</i> <b>ServerPort</b>	Erstellt ein Objekt der ServerApi mit Verbindungsparametern & verbindet die API automatisch mit dem Server.
AufgabenLibrary.AufgabenApi. <b>Connect()</b>		Verbindet die API manuell mit dem Server. <i>AufgabenApi.Host</i> & <i>AufgabenApi.Port</i> müssen gesetzt sein.
AufgabenLibrary.AufgabenApi. <b>Login()</b>	<i>string</i> <b>Username</b> , <i>string</i> <b>Passwort</b>	Loggt den übergeben Benutzer über die API ein.
AufgabenLibrary.AufgabenApi. <b>DomainLogin()</b>		Loggt den Benutzer anhand des Namens innerhalb der Domain ein.
AufgabenLibrary.AufgabenApi. <b>GetAufgaben()</b>	<i>int</i> <b>GRUPPEN_ID</b> , <i>SubNodeType</i> <b>subNodeType</b> , <i>bool</i> <b>nurNeueAufgaben</b> = false	Gibt eine DataTable mit den Aufgaben der gewählten Gruppe & des Gewählten SubNodes zurück. *
AufgabenLibrary.AufgabenApi. <b>GetAufgaben()</b>	<i>NodeType</i> <b>NodeType</b> , <i>SubNodeType</i> <b>subNodeType</b> , <i>bool</i> <b>nurNeueAufgab</b> = false	Gibt eine DataTable mit den Aufgaben des gewählten Nodes & des Gewählten SubNodes zurück. *
AufgabenLibrary.AufgabenApi. <b>GetAufgabe()</b>	<i>long</i> <b>A_NR</b> , <i>bool</i> <b>istAktiveAufgabe</b> = true	Gibt die <i>AufgabenLibrary.Library.Aufgabe</i> mit der angegeben ID zurück.
AufgabenLibrary.AufgabenApi. <b>CreateAufgabe()</b>	<i>AufgabenLibrary.Library</i> <b>.Aufgabe</b>	Erstellt anhand der im Parameter übergebenen Aufgabe einen neue Aufgabe auf dem Server.
AufgabenLibrary.AufgabenApi. <b>UpdateAufgabe()</b>	<i>AufgabenLibrary.Library</i> <b>.Aufgabe</b>	Aktualisiert die übergebene Aufgabe anhand der in ihr definierten Werte.

\*Nodes entsprechen den Kategorien wie "für mich / von mir" etc., SubNodes sind die Unterpunkte wie "inkl. Zukünftige"

**Achtung der NodeType Arbeitsgruppe darf nicht für das Abfragen der Aufgaben benutzt werden.**

## Eigenschaften :

<b>Datentyp</b>	<b>Eigenschaft</b>	<b>Zweck</b>
string	AufgabenLibrary.AufgabenApi. <b>Host</b>	Ruft den Host des Server ab, oder legt diesen fest.
int	AufgabenLibrary.AufgabenApi. <b>Port</b>	Ruft den Host des Port ab, oder legt diesen fest.

bool	<code>AufgabenLibrary.AufgabenApi</code> <code>.IsConnected</code>	Gibt an ob die API mit dem Server verbunden ist.
bool	<code>AufgabenLibrary.AufgabenApi</code> <code>.IsLoggedIn</code>	Gibt an ob die API mit dem Ausgewählten Nutzer am Server eingeloggt ist.
bool	<code>AufgabenLibrary.AufgabenApi</code> <code>.DieserMitarbeiter.IsAdmin</code>	Ruft einen Wert ab, der angibt, ob der Mitarbeiter ein Admin ist.
List<int>	<code>AufgabenLibrary.AufgabenApi</code> <code>.DieserMitarbeiter.IstGruppenChef</code> <code>Von</code>	Liefert alle GruppenIDs in denen dieser Mitarbeiter Gruppen-Chef ist.

## Verbindung zum Server aufbauen

### Verbindung mit dem AufgabenServer

Nutzen Sie für das Verbinden mit dem Server die erstellte Instanz der Aufgaben API, die 2 Möglichen Aufrufe sehen wie folgt aus :

#### 1) Server Daten im Konstruktor :

```
try
{
    aufgabenApi = new AufgabenApi("localhost", 5698);
}
catch (AufgabenLibrary.Exceptions.ConnectionRefusedException ex)
{
    MessageBox.Show("Verbindung Fehlgeschlagen", "Port wird vom Server abgelehnt",
"OK");
}
catch (AufgabenLibrary.Exceptions.InvalidVersionException ex)
{
    DisplayAlertInvoked("Version Veraltet", ex.Message, "OK");
}
}
```

Die Api verbindet sich automatisch mit dem Server.

#### 2) Server Daten nicht im Konstruktor :

```
AufgabenApi aufgabenApi = new AufgabenApi();
aufgabenApi.Host = "localhost";
aufgabenApi.Port = 5698;
```

```
bool connected = aufgabenApi.Connect();
```

Die Api verbindet sich **nicht** automatisch mit dem Server, sonder erst nach der Connect() Funktion.

### Login am AufgabenServer

Damit Sie auch Aufgaben an den Server senden können, müssen Sie sich zunächst am AufgabenServer einloggen.

Um sich am Server einzuloggen haben Sie zwei Möglichkeiten.

#### 1.) Manuelles Login

```
if (aufgabenApi.IsConnected)
{
    bool loggedIn = aufgabenApi.Login("MitarbeiterName", "MitarbeiterPasswort")
}
```

**2.) Domain Login** (Prüft ob der Name des Windowsnutzers in der Aufgabeplanung angelegt ist, falls ja wird mit diesem Nutzer verbunden.)

```
if (aufgabenApi.IsConnected)
{
    bool loggedIn = aufgabenApi.aufgabenApi.DomainLogin();
}
```

```
}
```

**Hinweis:**

Sobald Sie erfolgreich durch die API eingeloggt wurden, wird eine Client-Lizenz für die Dauer der Anmeldung genutzt. Sollten Sie zu wenige Lizenzen besitzen können Sie diese gerne bei uns bestellen unter

<https://www.karley.de/> oder per Email

E-Mail: [software@karley.eu](mailto:software@karley.eu)

oder Telefon: 02361-9792310

## Aufgabe(n) abfragen

Um Aufgaben abzufragen müssen Sie mit dem Server verbunden und eingeloggt sein.

**Einzelaufgabe abfragen:**

```
Aufgabe aufgabe = aufgabenApi.GetAufgabe(3);
```

*Hier wird die aktive Aufgabe mit der ID 3 abgefragt*

```
Aufgabe aufgabe = aufgabenApi.GetAufgabe(3,false);
```

*Hier wird die erledigte Aufgabe mit der ID 3 abgefragt*

**Aufgaben-Liste abfragen:** (Falls der Parameter nurNeueAufgaben auf **true** gesetzt wird, werden nur Aufgaben der letzten 8 Stunden angezeigt)

```
DataTable aufgabenTable = aufgabenApi.GetAufgaben(NodeType.FürMichVonMir,  
SubNodeType.KeinSubNode);
```

*//Hier werden alle Aufgabe abgefragt die im Reiter für mich / von mir angezeigt werden*

```
DataTable aufgabenTable = aufgabenApi.GetAufgaben(NodeType.FürMichVonMir,  
SubNodeType.Erledigte);
```

*//Hier werden alle Aufgabe abgefragt die im Reiter für mich / von mir - Erledigte angezeigt werden*

```
DataTable aufgabenTable = aufgabenApi.GetAufgaben(2, SubNodeType.Erledigte);
```

*//Hier werden alle Erledigten Aufgabe der Gruppe mit der ID 2 abgefragt.\**

*\*Gruppen können über die Eigenschaft Arbeitsgruppen abgefragt werden bsp.:*

```
aufgabenApi.Arbeitsgruppen[0].GRUPPEN_ID
```

## Neue Aufgabe anlegen/bearbeiten

Sobald Sie sich mit dem Server verbunden haben und sich auch erfolgreich eingeloggt haben, dann können Sie ein neues Aufgaben-Object erstellen und mit Daten füllen.

Fügen Sie das Aufgaben-Objekt nun zum Server-Protokoll hinzu mit der gewünschten Aktion. Im letzten Schritt müssen Sie das ServerProtokoll noch der NachrichtenQueue hinzufügen, diese versendet dann die Nachricht automatisch asynchron an den Server, welcher dann die Aufgabe anlegt.

Hier ein Beispielscript:

```
//neue Aufgabe anlegen
```

```
Aufgabe aufgabe = new Aufgabe();
```

```
aufgabe.ANLAGE_USERID = aufgabenApi.DieserMitarbeiter.USER_ID; //Erstellt vom  
eingeloggten Nutzer
```

```
aufgabe.AUFGABE_AN_G_ODER_U = "U"; //Aufgabe an User
```

```
aufgabe.AUFGABE_AN_ID = 2;
```

```
aufgabe.AUFGABE_DAUER = 5;
```

```
aufgabe.AUFGABE_FAELLIG = DateTime.Now.AddDays(4); //Fällig in 4 Tagen
```

```
aufgabe.AUFGABE_START = DateTime.Now;
```

```
aufgabe.BEARBEITET_USERID = 2; *
```

```
aufgabe.BESCHREIBUNG = "Aufgaben Beschreibungstext, plain Text kein RTF.";
```

```
aufgabe.BESCHREIBUNG_RTF = "Aufgaben Beschreibungstext - RTF"; // Falls angegeben muss
```

dieser Text in RTF Formatierung angegeben werden, dies ist hier zur Demonstration weggelassen worden.

```
aufgabe.TITEL = "Neue Aufgabe Titel";
aufgabe.WICHTIGKEIT = 3;
```

```
bool angelegt = aufgabenApi.CreateAufgabe(aufgabe);
```

Hier wurden nicht alle Werte der Aufgabe bearbeitet, auch andere Werte des Aufgaben-Objekts können bearbeitet werden, werden diese nicht bearbeitet, haben diese einen Default-Wert.

\*Eine Liste der Mitarbeiter erhält man über `GlobaleVariablen.Mitarbeiter` - `Dictionary<UserId,UserName>`

## Aufgabe Bearbeiten

Das Bearbeiten einer Aufgabe ist von der Logik, genau so aufgebaut wie das Erstellen einer Aufgabe.

```
//Bestehende Aufgabe holen
```

```
Aufgabe aufgabe = aufgabenApi.GetAufgabe(3);
```

```
aufgabe.AUFGABE_FAELLIG = aufgabe.AUFGABE_FAELLIG.AddDays(4); //Fällig 4 Tage hochsetzen
aufgabe.BEARBEITET_USERID = 4; //Aufgabe anderem Nutzer zuweisen.
```

```
bool gespeichert = aufgabenApi.UpdateAufgabe(aufgabe);
```

## Statusmeldungen der Funktionen

```
StatusMeldung result = aufgabenApi.CreateAufgabe(neueAufgabe);
```

```
if (!result)
```

```
{
```

```
    MessageBox.Show($"Aufgabe konnte nicht angelegt werden. Fehler:{result.Message}",
"Aufgabe konnte nicht angelegt werden", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Diese Statusmeldungen werden auch bei vielen Funktionen zurückgegeben, wie zB. bei den Funktionen zur Bearbeitung der Aufgaben.

## Klassenansicht

---

**AufgabenApi**  
Sealed Klasse

Felder

- ewh
- port

Eigenschaften

- Arbeitsgruppen
- Connection
- DieserMitarbeiter
- Host
- IsConnected
- IsLoggedIn
- Mitarbeiterliste
- Port
- Suche
- Zeiterfassung

Methoden

- ~AufgabenApi
- AufgabenApi (+ 1 Überladung)
- Connect
- CreateAufgabe
- DeleteAufgabe
- DomainLogin
- GetAufgabe
- GetAufgaben (+ 1 Überladung)
- GetMitarbeiterAufgaben
- InsertKommentar
- Login
- SendSql
- Srv\_KarussellAnfrageExpired
- UpdateAufgabe
- ZeiterfassungBeenden
- ZeiterfassungStarten
- ZeitNachtragen

**Aufgabe**  
Klasse

Felder

- \_AUFGABE\_AN\_G\_ODER\_U
- \_ERLEDIGT\_PROZENT
- \_WICHTIGKEIT
- \_WIEDERHOLUNG
- IstAktiveAufgabe
- MinBisdatum

Eigenschaften

- A\_NR
- ADRESSE
- ANLAGE\_DATUM
- ANLAGE\_USERID
- AUFGABE\_AN\_G\_ODER\_U
- AUFGABE\_AN\_ID
- AUFGABE\_DAUER
- AUFGABE\_EXISTIERT
- AUFGABE\_FAELLIG
- AUFGABE\_START
- AUFGABE\_WIRD\_ABGERECHNET
- BEARBEITET\_USERID
- BESCHREIBUNG
- BESCHREIBUNG\_RTF
- DENYCOUNT
- EMAIL\_WENN\_ERLEDIGT
- ERLEDIGT\_AM
- ERLEDIGT\_PROZENT
- GRUPPE
- INTERVAL\_TEIL1
- INTERVAL\_TEIL2
- INTERVAL\_TEIL3
- IST\_KUNDENAUFGABE
- KARUSSEL\_AUFGABE
- KOMMENTARE
- KOMMT\_VON\_TOBIT
- KUNDENNUMMER
- LASTCHANGE
- NICHT\_UEBERSPRINGEN
- PROJEKT\_ID
- PROJEKT\_NODE\_NAME
- PROJEKT\_START\_DISTANCE
- PROJEKT\_START\_FIX
- PROJEKT\_VORGAENGER\_ID
- TITEL
- WICHTIGKEIT
- WIEDERHOLUNG
- WIEDERHOLUNG\_BIS
- ZEIT\_GEBRAUCHT

Methoden

- Aufgabe (+ 1 Überladung)

**SubNodeType**  
Enumeration

- KeinSubNode
- Zukünftige
- Erledigte

**NodeType**  
Enumeration

- FürMichVonMir
- MeineAufgaben
- VonMirBeauftragt
- AbsolutAlle
- Arbeitsgruppe

**StatusMeldung**  
Struktur

Eigenschaften

- Message
- Status

Methoden

- implicit operat...

## Information zur API

Diese Dokumentation zeigt nur die aus unserer Sicht wichtigsten Dinge der API für den Endkunden. Auf Anfrage erweitern wir die Dokumentation gerne mit weiteren Beispielen.



Bei Fragen helfen wir gerne unter **[software@karley.eu](mailto:software@karley.eu)** weiter.