Home Automation - Weather Clock

Introduction

Author: Johan Lofstad, Microchip Technology Inc.

This user guide presents a *Weather Clock* design using the AVR® IoT WG Board. The weather clock fetches forecast data for a physical location and presents it to the user as a position on the clock. See Figure 1 for a photo of the clock on a sunny day.

Most of the design complexity resides in the cloud, and will be the focus of this user guide. The weather clock uses the Google Cloud Platform (GCP) as the cloud provider. Topics covered are:

- Fetching and storing weather data from yr.no
- Converting weather data to clock hand position
- Adding precise stepping functionality to the stepper motor driver on the AVR-IoT WG Board
- Automating the process using a cloud scheduler

Tip: It is recommended that the reader has read the preliminary *“Getting started with the AVR Home Automation Kit”*. It can be found at http://www.microchip.com/DS50002957.

Tip: The embedded source code for the weather clock can be found at https://start.atmel.com/#examples under the name “AVR IoT WG Sensor Node with Stepper 2 Click”. Under “Example Configuration”, select “WEATHER CLOCK”.

Figure 1. A Photo a Finished Weather Clock

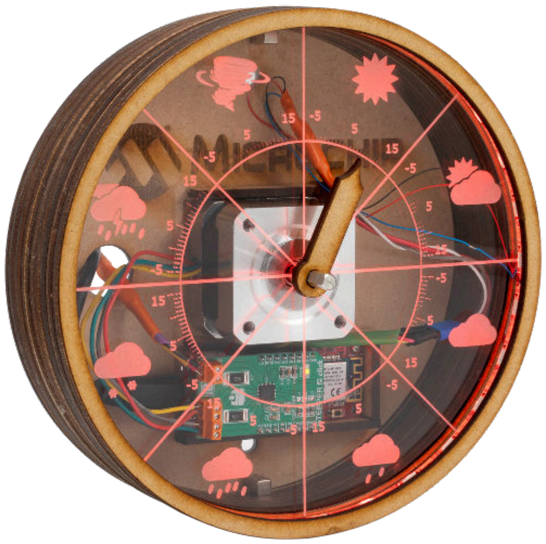


Table of Contents

1. Connecting Devices to the Cloud

Cloud modules used in this section

Cloud IoT Core

For IoT devices, the cloud is a remote platform that can be used to offload tasks such as computing, storage and communications. The big difference between a traditional server solution and the cloud is the *serverless* aspect. When working with cloud solutions, the servers and virtual machines are abstracted away. Instead, everything *lives* in the cloud, where the engineer only works on the solution without worrying about the underlying infrastructure.

The weather clock uses the **Google Cloud Platform**. At the time of writing, the free tier covers every requirement for the weather clock. The cloud service is configured through either its online *console* GUI or the command line argument utility *gcloud*. This user guide uses the online console GUI to configure all cloud modules.

The Google Cloud is divided into several modules based on your needs. The weather clock uses the modules given in Table 1-1. Each module has a specific purpose, allowing powerful functionality by making different modules interact.

Table 1-1. Cloud Modules Used by the Weather Clock

Module Name	Description	Documentation
IoT Core	Connect and manage IoT devices. Sends and Receives data, handles authentication.	https://cloud.google.com/iot-core/
Functions	Serverless functions which live in the cloud. Triggers based on events, such as an HTTP request or MQTT message.	https://cloud.google.com/functions/
Storage	Store and fetch files.	https://cloud.google.com/storage/
Scheduler	A fully managed cron job scheduler. Allows for automation in the cloud.	https://cloud.google.com/scheduler/

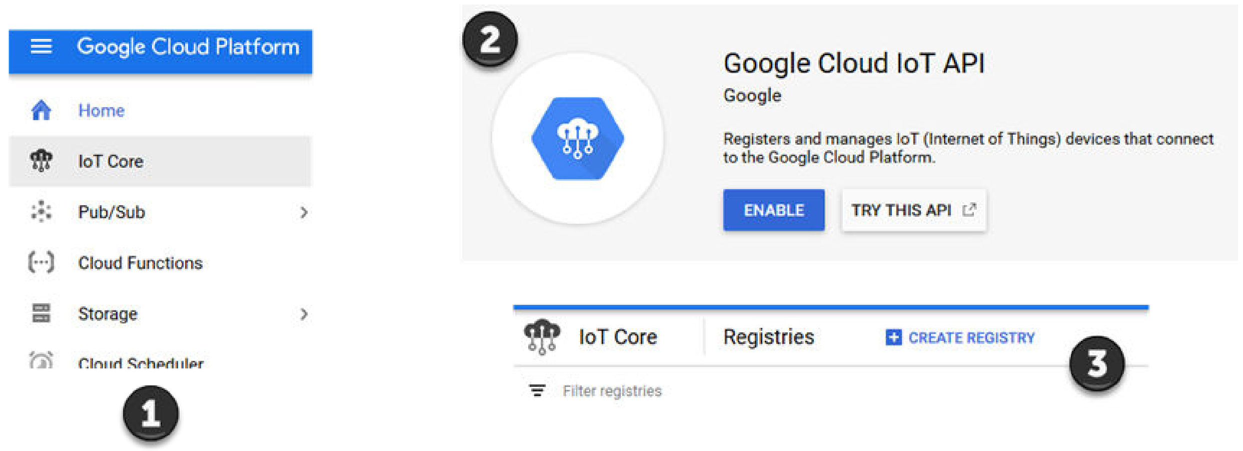
1.1 Configuring IoT Core

Tip: If you do not have a Google Cloud account, you can obtain one at https://console.cloud.google.com/freetrial.

The IoT Core module is designed to handle all communications with IoT devices. All devices are registered in the IoT Core with a unique ID and authentication credentials. In short, it serves as a *gateway* between the IoT devices and the rest of the cloud.

Google Cloud requires that all functionality belongs to a project. The weather clock uses the project name **iot-weather-clock**. The IoT Core module can be opened through the menu at the left-hand side of the cloud console. Click “Enable” to add the module to the cloud project. When the IoT Core is added, a *Registries* page should appear. See Figure 1-1 for screenshots of the procedure.

Figure 1-1. How to Find the IoT Core Module



To connect a device to the IoT Core Module, it must be added to a registry. A registry is a set of devices that can communicate with the cloud. To create a new registry, click the **Create Registry** button. There are several required fields. Configure the weather clock registry according to the "Entry" column in Table 1-2. Some fields might not appear before clicking "Show Advanced Options".

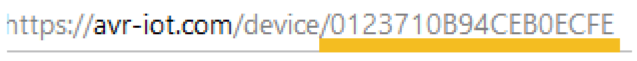
Table 1-2. IoT Core Create Registry Fields

Name	Entry	Description
Registry ID	weather-clock-devices	A permanent ID which identifies the registry
Region	The region which is applicable	The geographical region where the data are stored
Protocol	✓ MQTT ☐ HTTP	Which communications protocol does the registry support? Both MQTT and HTTP are supported.
Cloud Pub/Sub topics	Select the dropdown menu and select **Create a topic**. Enter the topic name *weather-upstream*. Leave the rest as default and press "Create topic".	The default telemetry topic is the MQTT topic, which all messages from the device are routed to.
Device state topic (optional)	Leave unchanged	All state events published by the device is sent there. Not used by the weather clock.
Stackdriver Logging	None	Not used by the weather clock

1.1.1 Adding a Device to the Registry

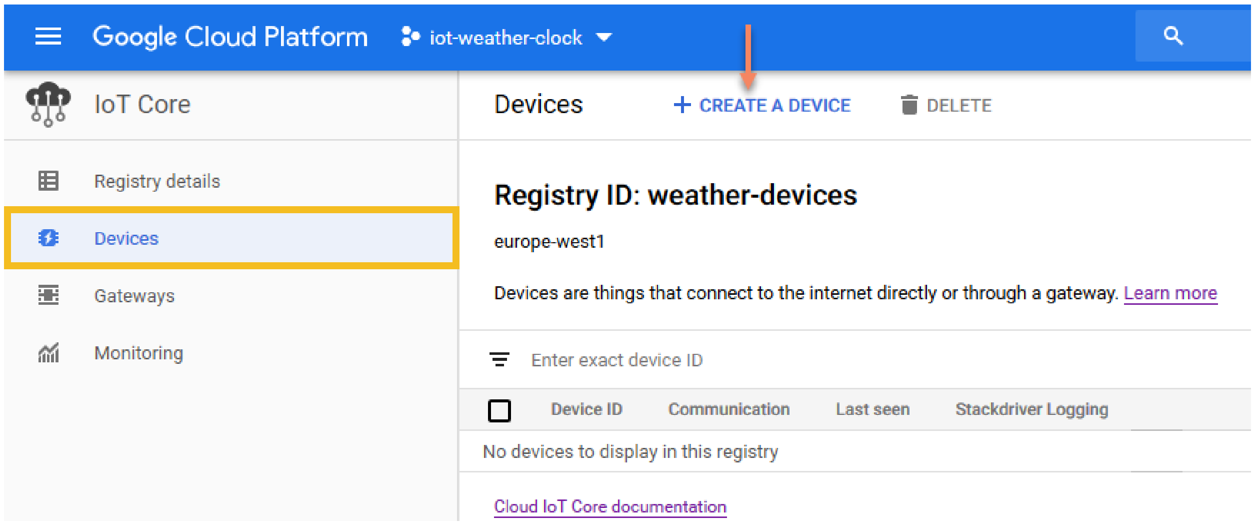
All devices in a registry are found by selecting the *devices* tab on the left-hand side. A new device can be added by pressing **Create a Device**. See Figure 1-3. To add the AVR IoT WG board, leave everything as default except "Device ID", "Public Key Format" and "Public key value". The *Device ID* is found in the URL of the "CLICK-ME.htm" file. The "CLICK-ME.html" file is located under the "CURIOSITY" drive when the kit is connected through USB, see the example in Figure 1-2. **Google Cloud requires the first character to be a letter. The entered Device ID should thus be "d + the number". For instance "d0123710B94CEB0ECFE".**

Figure 1-2. Finding the Device ID for the IoT Board



https://avr-iot.com/device/0123710B94CEB0ECFE

Figure 1-3. Adding a New Device to the IoT Core Registry



Registry ID: weather-devices

europe-west1

Devices are things that connect to the internet directly or through a gateway. Learn more

Enter exact device ID

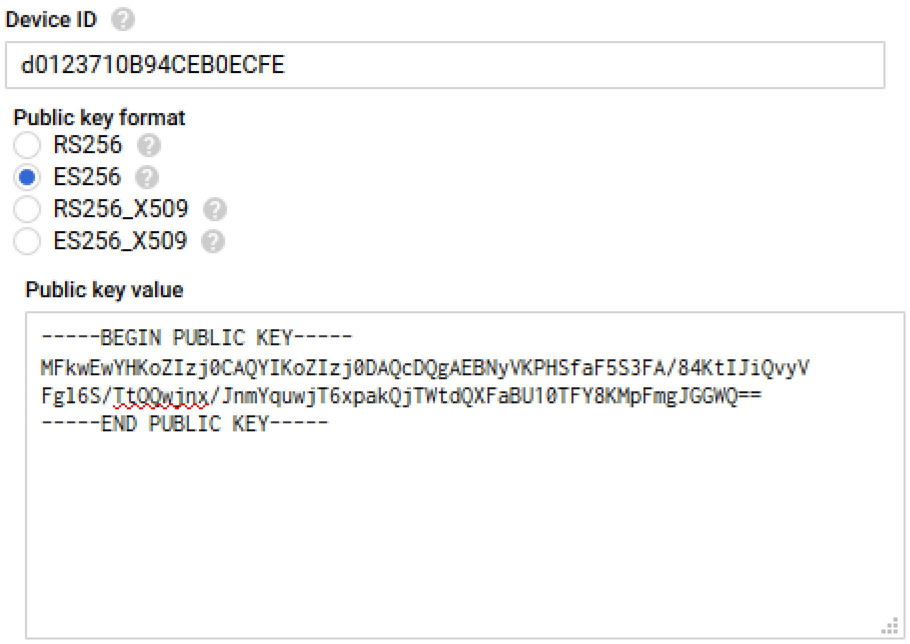
Device ID	Communication	Last seen	Stackdriver Logging

No devices to display in this registry

Cloud IoT Core documentation

The public key format is “ES256”. The public key is found in the “PUBKEY.txt” file under the CURIOSITY drive. Copy the contents in the public key value field. The details should be similar to Figure 1-4. Click “Create” to add the device.

Figure 1-4. Device Settings for the AVR® IoT WG Board



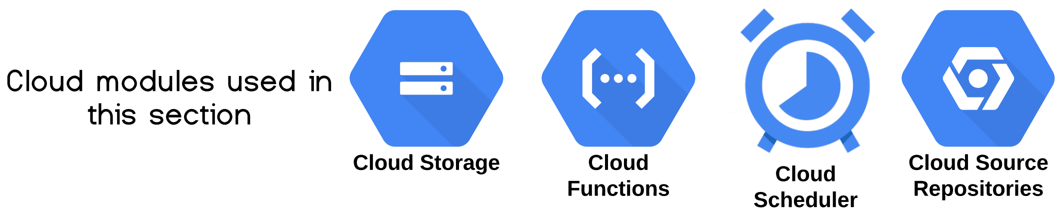
Device ID: d0123710B94CEB0ECFE

Public key format: RS256 [✓ ES256] RS256_X509 ES256_X509

Public key value

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEBNyVKPHSfaF5S3FA/84KtIJiQvyV
Fgl6S/ItQQwjnx/JnmYquwjT6xpakQjTWtdQXFaBU10TFY8KMpFmgJGGWQ==
-----END PUBLIC KEY-----
```

2. Fetching and Storing Weather Data

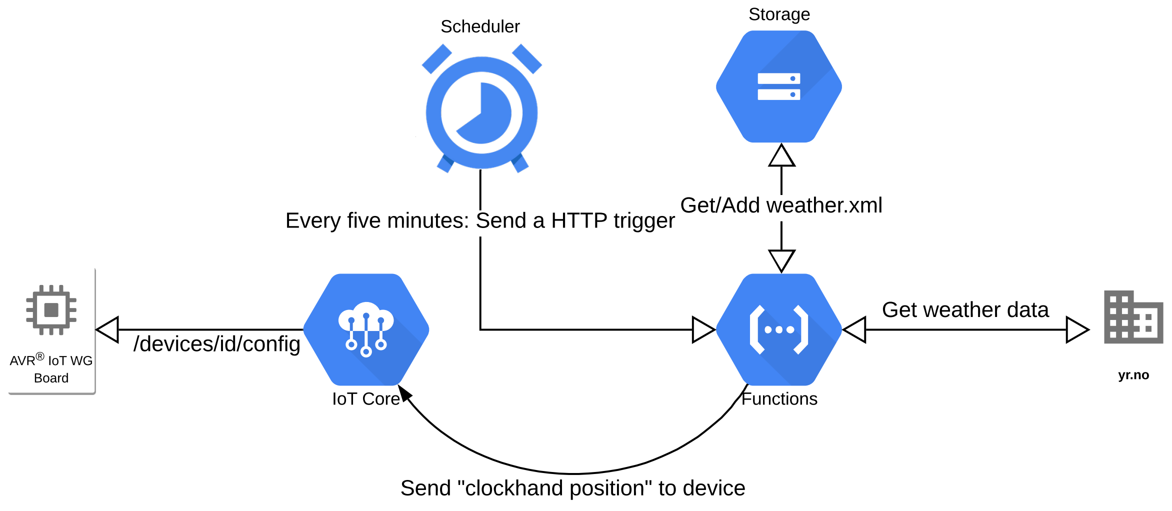


The weather clock requires up-to-date weather data to function. The data must be fetched, cached, parsed and converted to a clock hand position at a regular interval. Such a task is not handled by the device itself but by the cloud. This section explains how the cloud is configured and programmed to handle the given task. A block diagram of the finished solution can be seen in Figure 2-1.

The weather data is provided by YR.no, a free and open weather service from the Norwegian Meteorological Institute. At a regular interval, the weather data is downloaded from YR.no and stored using **Google Cloud Storage**. The downloading, parsing and converting is done by a python script through a **Cloud Function**. A cloud function is a snippet of code that is *serverless* (no dedicated resources, see Google's documentation), running when a *trigger* event occurs.

Tip: For more information on YR.no's API, see https://hjelp.yr.no/hc/en-us/articles/360001940793-Free-weather-data-service-from-Yr.

Figure 2-1. Block Diagram of the Finished Solution



2.1 Creating Persistent Storage

YR.no's API can only be accessed once every 30 minutes for one weather location. To handle this limitation, storage is introduced. As the cloud is serverless, some storage must be allocated to cache the data. Storage can be allocated through the *storage* module, creating *buckets* of data.

A bucket is created by navigating to the Storage module and clicking **Create bucket**. The bucket name is *globally unique* and is accessible by all projects on the platform (as long as they are authenticated). For the weather clock, every parameter can be left at their default values.

2.2 Uploading Source Code

A cloud function runs some source code whenever it is triggered. Source code can be uploaded to the Google Cloud through the "Source Repositories" module, which, in essence, is a git implementation. Google Cloud can connect to two external git providers, one of them being GitHub. The complete source code for processing the weather data can be found at https://github.com/microchip-pic-avr-solutions/avr-home-automation-weather-clock-cloud. By *forking* the repository to a personal GitHub account, it can be connected to gcloud, and device-specific configuration can be added. For more information on how to fork a GitHub repository, see GitHub.com.

A new source repository can be connected by navigating to the cloud source repositories module and clicking **Add Repository** in the upper right-hand corner, followed by *Connect external repository*. Select the project created earlier under "Project" and GitHub as the "Git provider". When selecting GitHub as the git provider, the forked repository should appear in the list. After selecting it and pressing **Connect selected repository**, it should appear as a repository in the cloud source repositories.

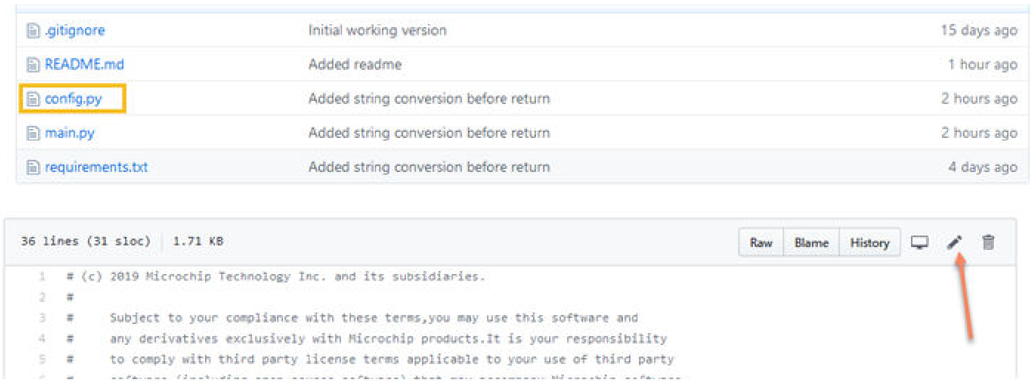
Tip: Git is a popular version control system. To learn more, see https://try.github.io/.

Tip: *(This tip assumes intermediate git knowledge.)* It is not necessary to connect to a GitHub account. Instead of "Connect external repository", a new repository can be created. Clone the provided GitHub repository, and change the remote on the cloned local repository. By using a force push, it can be pushed directly to the gcloud source repository.

2.3 Configuring the Source Code

Some configurations must be added to the source code for it to work properly with a given device. Navigate to the forked repository on GitHub, select the *config.py* file list followed by the *Edit Code* button (the pen icon). This allows the content of config.py to be modified. A screenshot can be seen in Figure 2-2.

Figure 2-2. How to Edit config.py



.gitignore	Initial working version	15 days ago
README.md	Added readme	1 hour ago
config.py	Added string conversion before return	2 hours ago
main.py	Added string conversion before return	2 hours ago
requirements.txt	Added string conversion before return	4 days ago

```
36 lines (31 sloc)  1.71 KB
# (c) 2019 Microchip Technology Inc. and its subsidiaries.
#
#     Subject to your compliance with these terms,you may use this software and
#     any derivatives exclusively with Microchip products.It is your responsibility
#     to comply with third party license terms applicable to your use of third party
```

There are eight fields to be configured. Each field is explained in Table 2-1. When the correct settings are entered, the changes can be applied by pressing the *Commit Changes* button.

Table 2-1. Source Code Config Fields

Name	Default entries	Description
PROJECT_ID	iot-weather-clock	The project ID of the gcloud project
IOT_CORE_REGION	europe-west1	The IoT Core region
IOT_CORE_REGISTRY_ID	weather-devices	The ID of the IoT Core configured in 1.1 Configuring IoT Core
IOT_CORE_DEVICE_ID	d0123710B94CEB0ECFE	The Device ID registered in 1.1.1 Adding a Device to the Registry
YR_LOCATION_URL	https://www.yr.no/place/United_Kingdom/England/London/forecast.xml	The YR.no API URL to fetch the weather data from. More info can be found at https://hjelp.yr.no/hc/en-us/articles/360009342913-XML-specification-of-forecast-xml.
CLOUD_STORAGE_BUCKET_ID	weather-clock-cache	The ID of the storage bucket created in 2.1 Creating Persistent Storage
TEMP_MAX	15	The temperature which maximizes the clock hand
TEMP_MIN	-5	The temperature which minimizes the clock hand

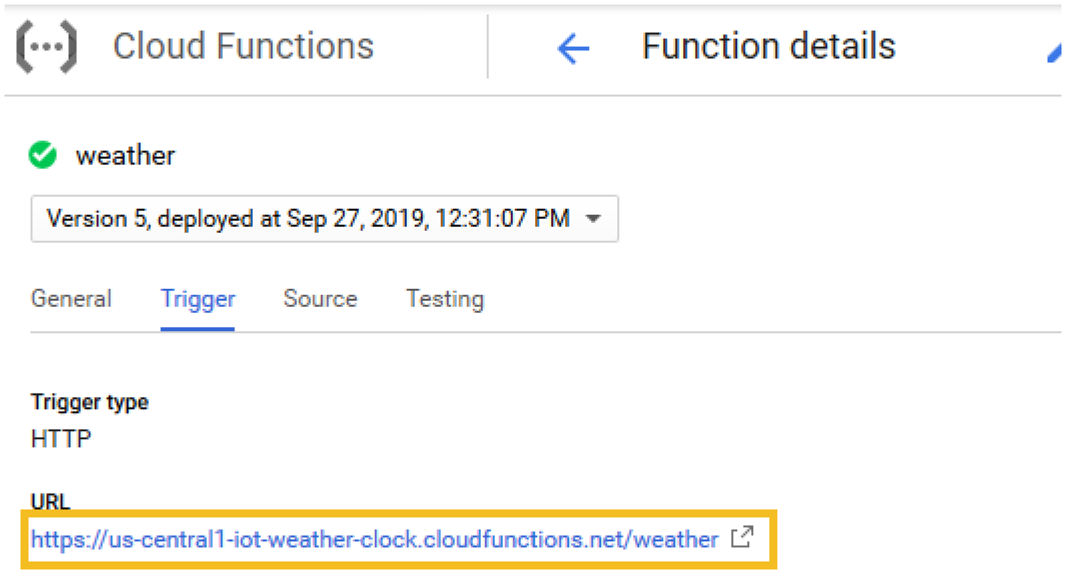
2.4 Creating a Cloud Function

A **Cloud Function** must be created to make the uploaded source code run at a given trigger. A cloud function is created by navigating to the Cloud Functions module and clicking **Create Function**. When creating a function, there are several required fields. Leave everything at default except for the fields shown in Table 2-2. When the cloud function has been created, it can be invoked by opening the *trigger URL*, found under the “trigger” tab of the newly created cloud function. A screenshot is shown in Figure 2-3.

Table 2-2. Cloud Function Fields

Name	Entry	Description
Name	weather	A name to identify the cloud function
Trigger	HTTP	Which event type should trigger the cloud function to run
Authentication	Unchecked: Disallow unauthenticated requests.	By not checking this box, triggers must be authenticated with the cloud for it to run
Source Code	Cloud Source repository	The location the source code is located
Runtime	Python 3.7	The programming language the code is written in
Repository	github_your_username_yr-weather-fetcher	The name of the repository the code is located at. This is the name of the repository created in 2.2 Uploading Source Code.
Function to excute	fetch_process_send	The entry point of the source code

Figure 2-3. The Cloud Functions *trigger URL* to Invoke Code Execution



weather

Version 5, deployed at Sep 27, 2019, 12:31:07 PM

Trigger type
HTTP

URL
https://us-central1-iot-weather-clock.cloudfunctions.net/weather

Tip: If changes are made to the source code, the changes are not propagated to the cloud function unless it is redeployed. To redeploy a cloud function, open it, select *edit* followed by *deploy*.

2.5 Running on a Schedule

To make the cloud fetch and process the weather data regularly, the **Cloud Scheduler** module is used to make the cloud function run on a thirty-minute interval. To create a new cloud scheduler *job* (task), navigate to the cloud scheduler module, and select *Create Job*. Each required field is described in Table 2-3. When the scheduler has been configured, the cloud function created in 2.4 Creating a Cloud Function should be called every 30 minutes, sending up-to-date weather data to the device.

Tip: If the reader is familiar with Linux® systems, one should find that the cloud scheduler in its simplest form is a cron job manager, a widely used scheduler found in many Linux systems.

Table 2-3. Cloud Scheduler Fields

Name	Entry	Description
Name	weather-scheduler	A unique name identifying the job
Frequency	*/30 * * * *	A schedule in the unix-cron format. The default entry runs every 30 minutes. See https://cloud.google.com/scheduler/docs/configuring/cron-job-schedules#defining_the_job_schedule for more information.

...........continued

Name	Entry	Description
Timezone	Greenwich Mean Time (GMT)	The timezone which the scheduler runs in. Depending on the frequency entry, the timezone can affect when it triggers.
Target	HTTP	When the scheduler triggers, it calls this target
URL	https://us-central1-iot-weather-clock.cloudfunctions.net/weather	The *trigger URL* of the cloud function created in 2.4 Creating a Cloud Function

2.6 Configuring Authentication

Important: This section is not necessary to understand to make the weather clock function. It is, however, recommended as a live setup would most likely require this level of security. If this section is skipped, anyone can invoke the cloud function and update the weather clocks position.

With the default settings, anyone with the trigger URL of the cloud function can start code execution. **Service Accounts** solve this authentication task and are Google's method of handling permissions in the cloud. A service account has specific permissions, such as *Invoke Cloud Function* and *Run Scheduler Job*. Cloud modules can be configured to only allow specific service accounts to access them. A thorough explanation of these service accounts can be found here: https://cloud.google.com/iam/docs/understanding-service-accounts.

To only allow the *weather-scheduler* job to invoke the *weather* cloud function, create a new service account. This is done by navigating to the "IAM & admin" module. Select "Service Accounts" on the left-hand side menu. Click "Create Service Account". The weather clock entries are shown in Table 2-4.

Table 2-4. Weather Service Account Fields

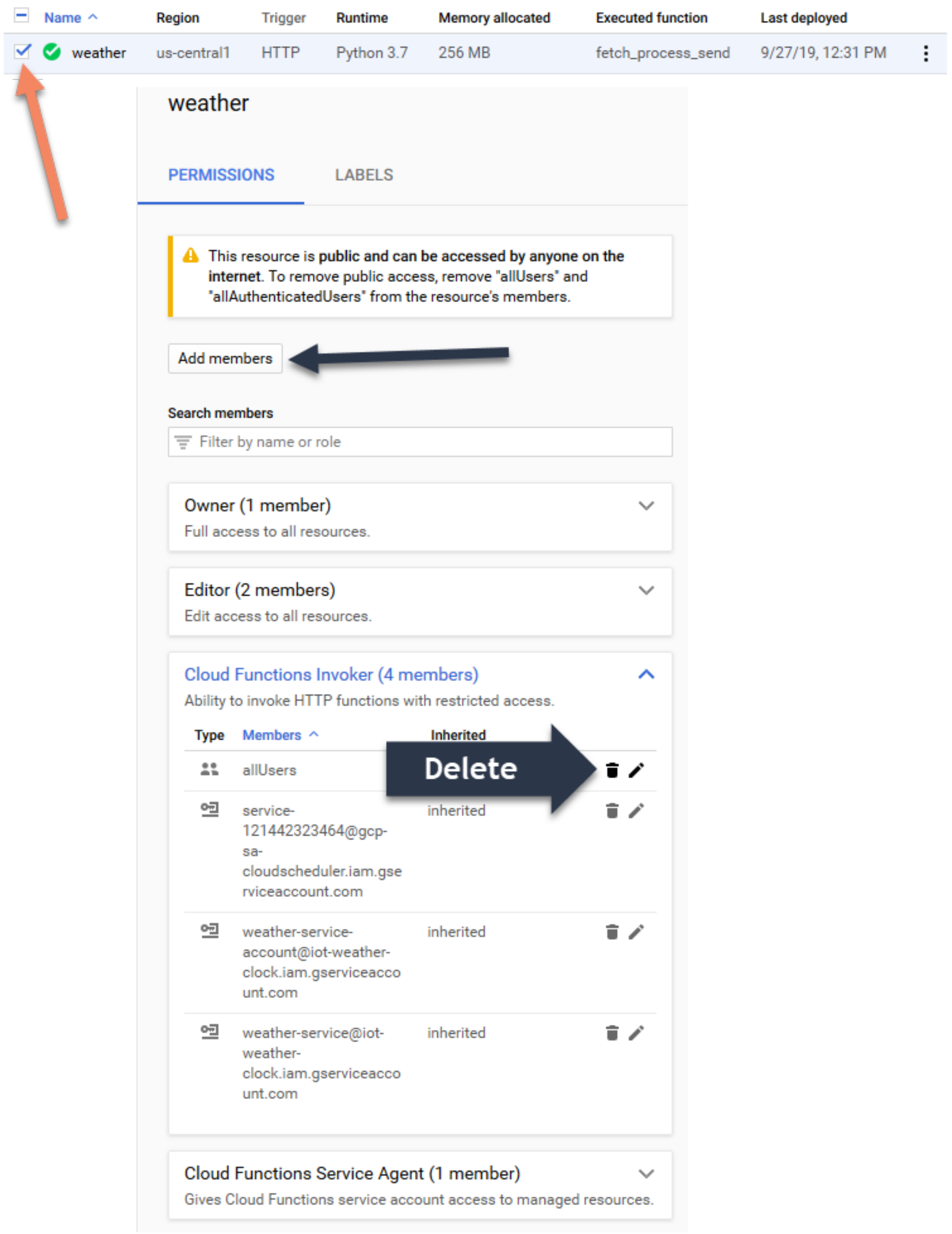
Name	Entry	Description
Service Account Name	weather-service	A display name for the service
Service account ID	weather-service	A unique identifier for the service account. Also known as the **Service Account Email**.
Service account description	Service account to invoke cloud functions with the cloud scheduler	-

When pressing continue, the cloud prompts for which *service account permissions* to add. The weather clock requires the following permissions.

- Cloud Scheduler Job Runner
- Cloud Functions Invoker

When the service account has been created, it appears in the service accounts table. The *email* field uniquely identifies each service account and is used to allocate permissions. The *cloud functions* and *cloud scheduler* modules must be configured to only allow the weather-service account to access them. At the cloud functions page, permission members can be added and removed by checking the checkbox to the left of the cloud function name, opening the permission panel on the right-hand side. See Figure 2-4 for a screenshot. By default, the *allUsers* group has permission to invoke. The *allUsers* permission is removed by clicking the trash can icon, removing the ability for anyone to invoke the function.

Figure 2-4. Cloud Function Permission Settings



Name	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed
weather	us-central1	HTTP	Python 3.7	256 MB	fetch_process_send	9/27/19, 12:31 PM

weather

PERMISSIONS LABELS

This resource is **public and can be accessed by anyone on the internet**. To remove public access, remove "allUsers" and "allAuthenticatedUsers" from the resource's members.

Add members

Search members

Filter by name or role

Owner (1 member)
Full access to all resources.

Editor (2 members)
Edit access to all resources.

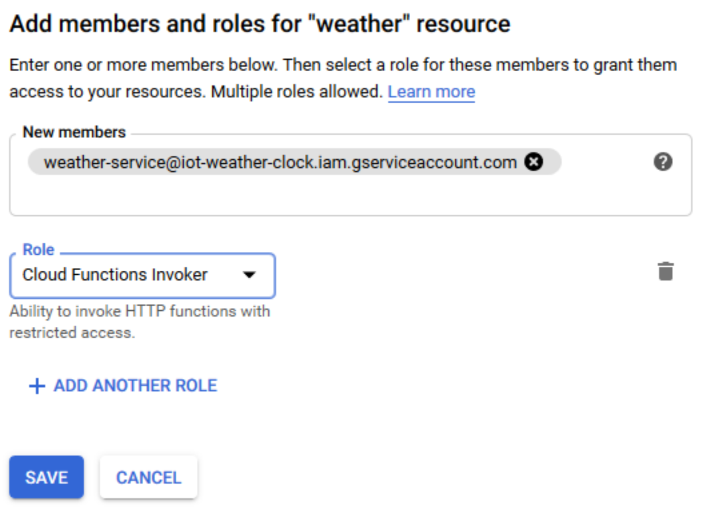
Cloud Functions Invoker (4 members)
Ability to invoke HTTP functions with restricted access.

Type	Members	Inherited
	allUsers	
	service-121442323464@gcp-sa-cloudscheduler.iam.gserviceaccount.com	inherited
	weather-service-account@iot-weather-clock.iam.gserviceaccount.com	inherited
	weather-service@iot-weather-clock.iam.gserviceaccount.com	inherited

Cloud Functions Service Agent (1 member)
Gives Cloud Functions service account access to managed resources.

To permit a service account to invoke the function, click the **Add Member** button, opening an *"Add Members"* dialog. The *"New Members"* field selects which service account to gain permission. Enter the service account email created in the previous step. By selecting the role to be *Cloud Functions Invoker*, the given service account will be given permissions to run the cloud function. See Figure 2-5 for a screenshot.

Figure 2-5. Adding Weather Service as a Member to the Weather Cloud Function


Add members and roles for "weather" resource

Enter one or more members below. Then select a role for these members to grant them access to your resources. Multiple roles allowed. Learn more

New members: weather-service@iot-weather-clock.iam.gserviceaccount.com

Role: Cloud Functions Invoker

Ability to invoke HTTP functions with restricted access.

With the above configuration, all HTTP triggers must be authenticated with the weather-service accounts credentials. The cloud scheduler must be configured to use these credentials to invoke the cloud function. These are added by editing the cloud scheduling job in the scheduler module and filling in the information given in Table 2-5 (press *Show More* to see these fields).

Table 2-5. Cloud Scheduler Authentication Fields

Name	Entry	Description
Auth header	Add OIDC Token	How the HTTP request is authenticated. For a service account, this is usually OIDC.
Service Account	*Example entry:*weather-service@iot-weather-clock.iam.gserviceaccount.com	Which service accounts credentials are sent. Must be a service account that has cloud function invoking rights.
Audience	*Example entry:* https://us-central1-iot-weather-clock.cloudfunctions.net/weather	Auto filled

3. Setting up the IoT Board

Important: The concepts and source code discussed in this section heavily rely on the source code discussed in the *Getting Started with the Home Automation Kit document*. It can be read here: http://www.microchip.com/DS50002957.

Important: The complete source code for the firmware can be found at https://start.atmel.com/#examples under the name “AVR IoT WG Sensor Node with Stepper 2 Click”. Under “Example Configuration”, select “WEATHER CLOCK”. If there is no desire to understand how the firmware and drivers function, this section can be skipped.

The IoT Board code from the Getting Started guide (http://www.microchip.com/DS50002957) must be modified slightly to become a weather clock. The following section expands the motor driver to do precise stepping and implements cloud communication to receive weather forecast data.

The starting point for the source which is expanded upon in this section can be obtained from Atmel START (https://start.atmel.com/#examples) under the name “AVR IoT WG Sensor Node with Stepper 2 Click”. Select “BASIC_MOTOR_CONTROL” under “Example Configuration”.

3.1 Expanding the Motor Driver: Precise Stepping

Precision is one of the major advantages of a stepper motor. For every step, the number of degrees rotated is known exactly. Due to this unique property, the motor can be controlled in an **open loop**. This implies that there is no need for any sensors to track the motor’s position, as the position can always be deducted as long as the starting position is known.

Every stepper motor has a **Step Angle** corresponding to the number of degrees the motor rotates for every pulse. The number of steps to send to move the motor by a given angle is found by Equation 3-1. The following code is a simple method to step precisely:

Equation 3-1. Number of Steps to Move the Motor a Given Angle

$$\#steps = \frac{angle}{STEP\_ANGLE}$$

```
uint16_t number_of_steps = degrees_to_step / STEP_ANGLE;
// For all steps, do one pulse
for(uint16_t i = 0; i < number_of_steps; i++){
    MOTOR_ST_toggle_level();
    _delay_ms(1);
    MOTOR_ST_toggle_level();
    _delay_ms(1);
}
```

The Adafruit Stepper Motor found in the Home Automation Kit has a step angle of $1.8°$, giving $\frac{360}{1.8} = 200$ possible positions for the stepper motor. The weather clock receives a clock hand position between zero and 200. Depending on the current position of the clock hand, the motor must rotate either clockwise (CW) or counter-clockwise (CCW). The motor should rotate in the direction with the shortest distance. The two distances can be calculated as shown in Equation 3-2.

Equation 3-2. Distance Calculations for Clock Hand Travel

$$CCW = (current\_position - target\_position)\ \%\ 200$$

$$CW = (target\_position - current\_position)\ \%\ 200$$

The percentage sign (%) implies a modulo operation, as a circle operates with 360 modular arithmetic. However, there are only 200 possible states, hence modulo 200 instead of 360. For instance, the summation $190 + 20 \rightarrow 10$.

Any position above 200 can be converted to the equivalent position under 200 by a modulo 200 operation ($position \% 200$).

When the shortest distance in both the CW and CCW direction has been calculated, the motor rotates the appropriate number of degrees by the shortest path. The current position is updated when the motor has rotated. The following code implements this:

```
        // Calculate clockwise and counter clockwise distance
    int16_t counter_clockwise = MATH_MODULO(current_position - target_position, 200);
    int16_t clockwise = MATH_MODULO(target_position - current_position, 200);

    // How many steps (and in which direction) do we have to turn to reach target_position?
    uint16_t number_of_steps = 0;
    if(clockwise < counter_clockwise){
        motor_set_direction(MOTOR_DIRECTION_CLOCKWISE);
        number_of_steps = clockwise;
        current_position = MATH_MODULO(current_position + number_of_steps, 200);
    }else{
        motor_set_direction(MOTOR_DIRECTION_COUNTER_CLOCKWISE);
        number_of_steps = counter_clockwise;
        current_position = MATH_MODULO(current_position - number_of_steps, 200);
    }
```

Tip: MATH_MODULO refers to using the mathematical definition of modulo and not the C version of modulo, which is different. The exact definition of MATH_MODULO is #define MATH_MODULO(n,M) (((n % M) + M) % M).

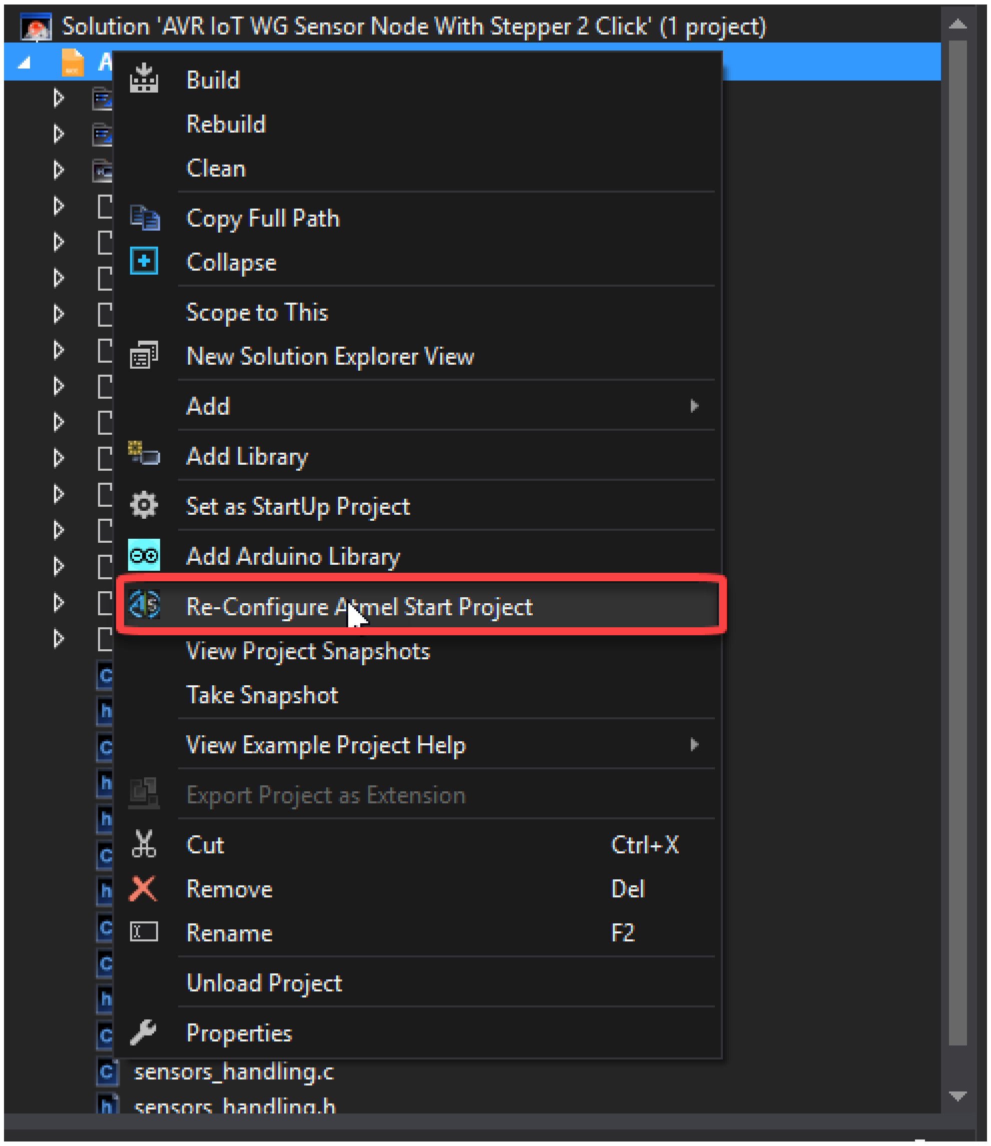
3.2 Cloud Communication

As weather data becomes available, the cloud configuration from 2. Fetching and Storing Weather Data sends a *position* MQTT message to the board, which represents the clock hand position. The handling and decoding of this "*position*" message are the same as the *speed* messages in the Getting Started Guide (http://www.microchip.com/DS50002957) but calling the *motor_goto_position(position)* instead.

```
if(*endptr == '\0'){
    // Successful conversion of the entire string. Set position
    debug_printer(SEVERITY_NONE, LEVEL_NORMAL, "Setting position %d \n", position);
    motor_goto_position(position);
}
```

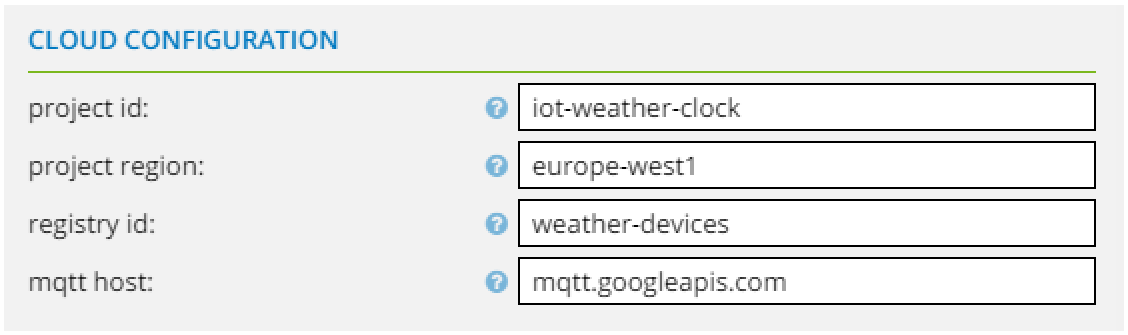
By default, the firmware is linked to the example project in Google Cloud and must be changed to the cloud project created earlier in this user guide. This is done by reconfiguring the Atmel START project. In Atmel Studio, right-click the project in the solution explorer. Select "Re-Configure Atmel Start Project". See Figure 3-1 for a screenshot.

Figure 3-1. Reconfiguring Atmel START



The settings for which cloud project to connect to can be changed by clicking the AVR IoT WG Sensor Node box. All the details can be entered in the “Cloud Configuration” table. Edit Project ID, Project Region, and Registry ID to the respective values from section 1. Connecting Devices to the Cloud. Leave MQTT Host at default. See Figure 3-2 for a screenshot. When clicking “Generate Project”, these new settings are saved to the source code.

Figure 3-2. Cloud Configuration in Atmel START



CLOUD CONFIGURATION

project id:	iot-weather-clock
project region:	europe-west1
registry id:	weather-devices
mqtt host:	mqtt.googleapis.com

4. Revision History

Doc. Rev.	Date	Comments
A	02/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-5621-6

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
http://www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra’anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820