TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics and Interdisciplinary Studies

Remote task management in embedded devices

Master Thesis

Study programme:	N0714A150003 Mechatronics
Author:	**Bc. Nikita Nagornov**
Thesis Supervisors:	Ing. Jan Kraus, Ph.D. Institute of Mechatronics and Computer Engineering
Thesis Consultant:	Ing. Ekaterina Nyrobtseva Institute of Mechatronics and Computer Engineering

Liberec 2020

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics and Interdisciplinary Studies

Master Thesis Assignment Form

Remote task management in embedded devices

Name and surname:	**Bc. Nikita Nagornov**
Identification number:	M19000212
Study programme:	N0714A150003 Mechatronics
Assigning department:	Institute of Mechatronics and Computer Engineering
Academic year:	**2019/2020**

Rules for Elaboration:

1. Study and analyse all requirements for a flexible task scheduler system with user friendly management of large number of tasks for data acquisition, processing and export in a multi-user environment.
2. Design the required infrastructure and develop your own application and its interfaces to allow simple handling of multiple automated tasks for a given embedded Linux environment.
3. Check the propper function of your application and evaluate its key performance indicators under real conditions.
4. Make a brief summary of your achievements in conclusion, discuss most important advantages and disadvantages of the presented solution and evaluate possibilities for its further improvements.

Scope of Graphic Work:	by appropriate documentation
Scope of Report:	40–50 pages
Thesis Form:	printed/electronic
Thesis Language:	English

List of Specialised Literature:

[1] KELLER, Michael S. Take command: cron: Job scheduler. Linux Journal, 1999, 1999.65es: 15.
[2] ENVIS Application User Guide: Version 1.8 for Supported Measuring Instruments [online]. Liberec: K M B systems, 2019 [cit. 2019-11-22].
[3] SAUTER, Thilo; LOBASHOV, Maksim. End-to-end communication architecture for smart grids. IEEE
[4] Transactions on Industrial Electronics, 2011, 58.4: 1218-1228.
[5] HILLAR, Gaston C. MQTT Essentials-A Lightweight IoT Protocol. Packt Publishing Ltd, 2017.
[6] PENG, Ya-Shiang; CHEN, Yen-Cheng. SNMP-based monitoring of heterogeneous virtual infrastructure in clouds. In: 2011 13th Asia-Pacific Network Operations and Management Symposium. IEEE, 2011. p. 1-6.

Thesis Supervisors:	Ing. Jan Kraus, Ph.D. Institute of Mechatronics and Computer Engineering
Thesis Consultant:	Ing. Ekaterina Nyrobtseva Institute of Mechatronics and Computer Engineering
Date of Thesis Assignment:	November 8, 2019
Date of Thesis Submission:	May 18, 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
Dean

doc. Ing. Milan Kolář, CSc.
head of institute

Liberec November 8, 2019

Declaration

May 30, 2020 Bc. Nikita Nagornov

Vzdálená správa úloh ve vestavěných zařízeních

Abstrakt

Práce obsahuje výzkum nezbytný k vytvoření kompletního řešení pro správu a zpracování informací získaných z elektroměrů. Zabývá se serverem používaným pro připojení k zařízením, nastavováním úkolů a jejich správou na každém zařízení.

Klíčová slova: Internet věcí, plánovač úloh, vestavěné systémy, plánovač úloh, colibri, monitorovací server

Remote task management in embedded devices

Abstract

The work contains the research necessary to create a complete solution for the management and processing of information obtained from electric meters. It deals with a server used to connect to devices, set tasks, and manage them on each device.

Keywords: Internet of Things, task scheduler, embedded systems, jobscheduler, colibri, monitoring server

Acknowledgements

I would like to express my deep and sincere gratitude to my thesis supervisor Ing. Jan Kraus, Ph.D. at the Technical University of Liberec. Prof. Kraus was always open whenever I had a trouble spot or a question about my research or writing. He consistently allowed this paper to be my work but steered me in the right direction whenever he thought I needed it.

I would like to thank the experts who were involved in the validation survey for this research project: Ing. Tomas Bedrnik and Ing. Ekaterina Nyrobtseva. Without their passionate participation and input, the work could not have been successfully conducted.

I would also like to thank the Ing. Jaroslav Nosek, CSc. with the leaderships of the Technical University of Liberec and Erasmus for the excellent opportunity to live and study in the progressive and fast developing university.

Contents

List of abbreviations

IoT	Internet of Things
LAN	Local Area Network
PAN	Personal Area Network
WAN	Wide Area Network
GSM	Global System for Mobile Communications
GPRS	General Packet Radio Service
LTE	Long-Term Evolution
WSN	Wireless Sensor Network
WoT	Web of Things
HTTP	HypperText Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
USB	Universal Serial Bus
NAS	Network Attached Storage
RAID	Redundant Array of Independent Disks
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ARM	Advanced RISC Machine
RISC	Reduced Instruction Set Computer
RAM	Random-access Memory
LCD	Liquid-crystal Display
RGB	Red, Green, Blue
OTG	On-The-Go
SDIO	Secure Digital Input Output
CAN	Controller Area Network
PWM	Pulse Width Modulation
SQL	Structured Query Language
PL	Procedural Language
MMC	Microsoft Management Console
JOC	JobScheduler Operations Center
XML	Extensible Markup Language
LIFO	Last in, First out
FIFO	First in, First out
HTTP	Hypertext Transfer Protocol
DBMS	Database Management System

Introduction

The work is devoted to the development and implementation of a software package for organizing, managing, and monitoring the automated execution of various tasks on embedded devices.

The work is of interest for the reason that devices running on the Internet of things are autonomous. They must perform many scheduled tasks without the intervention of the end-user: collecting data from sensors, their primary processing to provide them in a way that is understandable to the user, logging, exchanging these data with other network devices, and transferring them to administrators for analysis. Business processes in organizations using IoT devices are not permanent. They may undergo some changes, which in turn may necessitate adjustments to the tasks performed by devices autonomously. Making changes to the software running on the device is difficult. In addition to the time spent on updating the software, it takes time to install new versions of programs on the devices themselves, which can lead to disabling and flashing them, which is impossible in most situations. Therefore, it is preferable to use a flexible task management system on devices with the ability to remotely create, set a schedule for their implementation, monitor the status and results of these tasks.

On the Internet of things, as a rule, many devices work, so working with each individual is not always convenient. For this reason, it is advisable to develop a centralized server that can manage and monitor the status of tasks on multiple devices on the network. Thus, the development and implementation of a software package for organizing centralized management of tasks on the Internet of things devices are set as a task. The following steps are taken to solve the problem: To be able to create tasks on the devices themselves, it is proposed to use the existing task scheduler on the market. There are several such planners. Some are components of embedded operating systems installed on devices; others are created by third-party developers and are universal solutions. It is necessary to analyze the existing implementations of the task planners and select the solution that is most suitable for the project within which this work is carried out.

For centralized remote task management on network devices, it is planned to develop their solution, a server that must connect to end devices, issue commands for managing tasks, and also poll information about created tasks. In turn, the server itself provides a REST API web interface through which clients can interact with the server and manage tasks on the devices to which the server is connected. Based on the provided REST API server, a client web application for IoT network administrators can be created.

1 Review of the Internet of Things

1.1 IoT Overview

The Internet of Things (IoT) hangs on three basic principles. Firstly, the ubiquitous communication infrastructure, secondly, the global identification of each object and, thirdly, the ability of each object to send and receive data via a personal network or the Internet to which it is connected. In IoT, each thing has its identifier, which together forms a continuum of things that can interact with each other, creating temporary or permanent networks. So things can take part in the process of moving them, sharing information about the current geo-location, which allows automating the logistics process, and having built-in intelligence thoroughly, things can change their properties and adapt to the environment. Internet of things has a single interaction protocol, according to which any network node is equal in the provision of its services. Each node of the Internet of things network provides its service, providing a specific data delivery service. At the same time, a node in such a network can receive commands from any other node, which leads all Internet of Things can interact with each other and solve joint computing problems. The Internet of Things can form local networks, united by any one service area [1].

The Internet of things conceptually belongs to next-generation networks, and consists of a set of different information and communication technologies that enable the functioning, and its architecture shows how these technologies are related to each other. The IoT architecture includes four functional layers (Figure 1.1), described below.

1.1.1 Sensors and Sensor Network Level

The lowest level of IoT architecture consists of “smart” objects integrated with sensors. Sensors realize the connection between the physical and digital worlds, providing the collection and processing of information in real-time. Miniaturization, which led to a reduction in the physical dimensions of hardware sensors, made it possible to integrate them directly into objects of the physical world. Most sensors require a connection to a sensor gateway, which can be implemented using a local area network (LAN), such as Ethernet, Wi-Fi, or a personal area network (PAN). For sensors that do not require a connection to the gateway, their communication with servers/applications can be provided using global wireless WANs, such as GSM, GPRS, and LTE. Sensors, which are characterized by low power consumption and

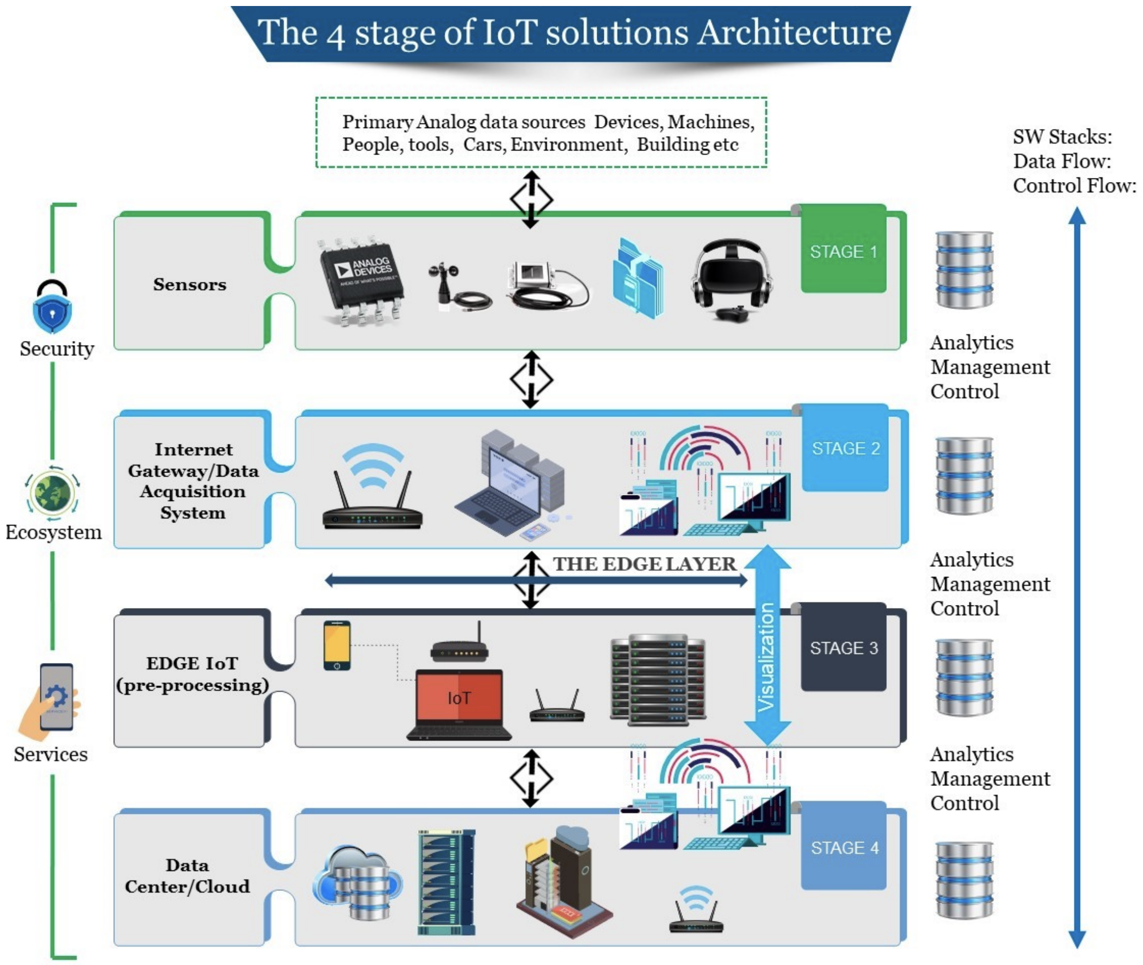


Figure 1.1: Internet of Things Architecture [2]

low data transfer rate, form the well-known WSN wireless sensor networks.

1.1.2 Gateway and Network Level

The large amount of data provided by the multiple sensors on the first IoT layer requires a reliable and high-performance wired or wireless infrastructure as a transport medium. This level consists of converged network infrastructure, which is created by integrating heterogeneous networks into a single network platform. The service level contains a set of information services that automate technological business operations in IoT, supporting operational and business activities, various analytical information processing, data storage, information security, business process management.

1.1.3 Application Level

At this level, there are various types of applications for the relevant industrial sectors and fields of activity (energy, transport, trade, medicine, education). Applications can be “vertical” when they are “specific” for a particular industry, as well as “horizontal”, which can be used in various sectors of the economy. An integral part of the Internet of things is the Web of Things (WoT), which enables the interaction of various intellectual objects (“things”) using Internet standards and mechanisms. For example, a URI (Uniform Resource Identifier), HTTP (HypperText Transfer Protocol), the style of building a distributed application architecture REST (Representational State Transfer). WoT provides for the implementation of the IoT concept at the application level using existing architectural solutions focused on the development of web applications. There are three methods of interacting with the Internet of things: direct access, access through a gateway, and access through a server. In the case of direct access, all devices must have their IP address or network alias, by which any client application can access them. The interface usually in the form of a web-site with a graphical interface for managing via a web browser. It is possible to use specialized software. In such web devices, the REST API should be integrated for direct access to them via the Internet. Access to the Internet of things through a gateway is a more rational way of organizing interaction. It completely displaces the direct access method, if necessary, to establish a connection between wireless sensor networks or the Internet of things network with the global Internet network. Most standards of wireless sensor networks do not support the IP protocol, using their interaction protocols, this feature makes it necessary to have a device for relaying messages from the sensor network to the Internet for protocol compatibility. The third form of device interaction in IoT through the server implies the existence of an intermediary between devices and the user. It can be implemented using an intermediary data platform. This approach assumes the presence of a centralized server or group of servers, the main functions of which include [3]:

- Receiving messages from the Internet of Things and transmitting them to users.
- Storage accepted and its processing).
- Providing a user interface with the possibility of a two-way exchange between the user and the Internet of thing.

The classical architecture of the Internet of things includes:

- IoT devices. They collect readings from sensors and perform physical actions. Can be personalized, wearable and integrated.
- Gateways that receive information from devices and send them commands to perform actions. Typically represented by a hardware router or software.
- Use different protocols.
- The server where the sensor readings are stored, processed, and analyzed. It can be implemented based on a virtual server, a real machine, or through the cloud.

- The client part is implemented through a mobile or web application, provides access to data and visualization of analysis results.

In general, the scheme of such a service is presented in Figure 1.2.

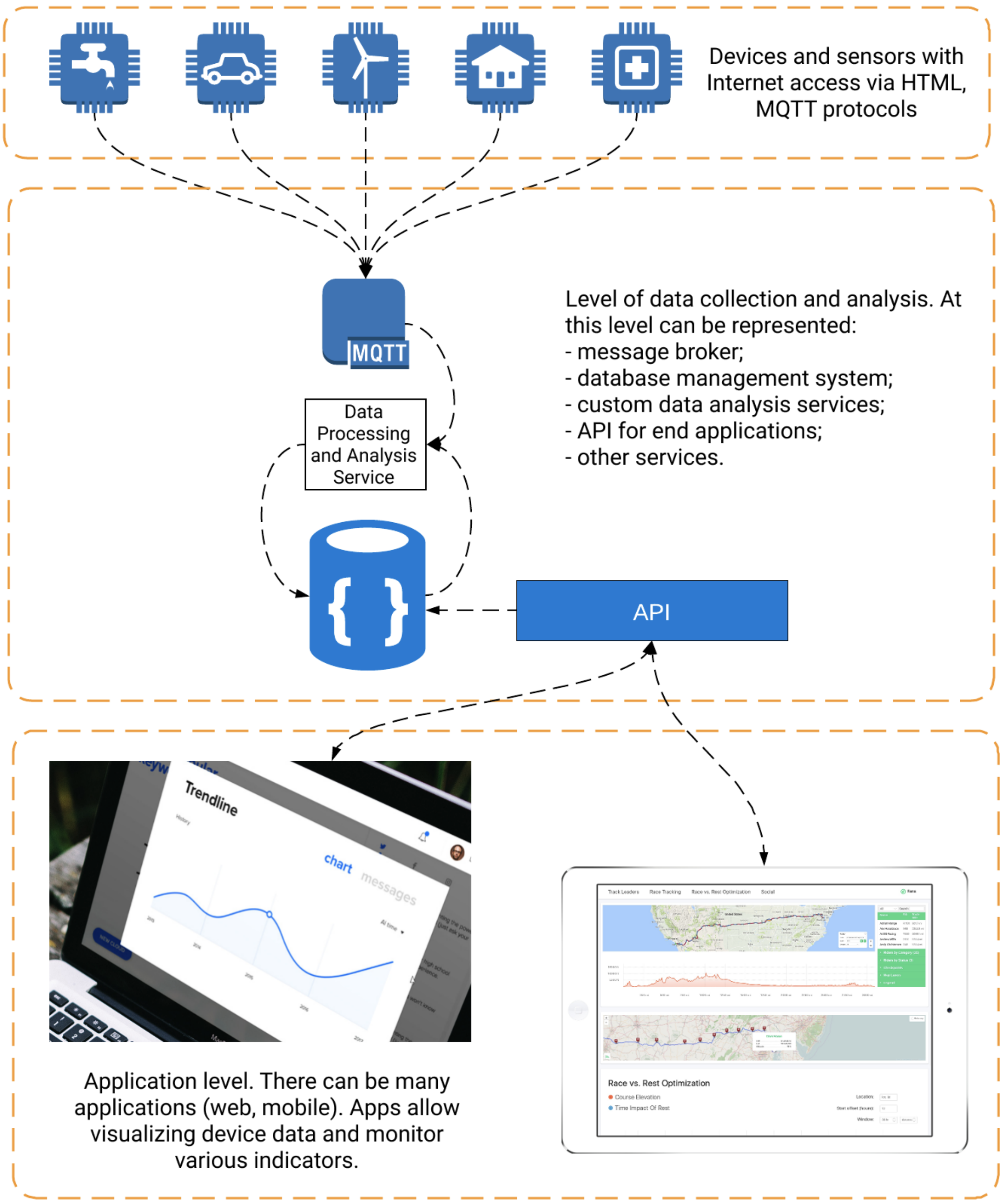

Devices and sensors with Internet access via HTML, MQTT protocols
MQTT
Data Processing and Analysis Service
Level of data collection and analysis. At this level can be represented:
- message broker;
- database management system;
- custom data analysis services;
- API for end applications;
- other services.
API
Trendline
chart
messages
Application level. There can be many applications (web, mobile). Apps allow visualizing device data and monitor various indicators.
Race vs. Rest Optimization

Figure 1.2: Scheme of the IoT service [4]

Thus, the idea of the Internet of things in the interaction of things with the server and with each other, where human participation is minimized. For example, electricity meters that send testimonies to a management company, trackers that

Table 1.1: Comparison of the characteristics of open source systems [5, 6, 7, 8, 9]

Title	Support						
	Open source code	Open hard-ware	Open source for drivers	Windows support	Modi-fied compo-nents	Manu-factory updates	Long term availa-bility
Raspberi Pi	Yes	No	No	Yes	Yes	Yes	Yes
ZoneMinder	Yes	No	No	No	No	No	No
Orange Pi	Yes	Yes	Yes	No	Yes	Yes	No
Colibri	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Rock Pi	Yes	Yes	Yes	No	Yes	Yes	Yes

track the movement of taxis, a variety of fitness bracelets — all this is the Internet of things.

1.1.4 Overview and Analysis of Single-board Computer Modules

Single-board computer modules have been manufactured for a long time. However, as soon as the Raspberry Pi platform appeared, these devices gained immense popularity. To select the best platform for the price-quality ratio, consider the popular models (Table 1.1).

The Raspberry Pi is an incredibly popular device known for its affordability, versatility, capabilities, and vibrant community. When the first model appeared in 2012, it became an essential milestone in the single-board market. Although several good boards already existed, such as Beagleboard and Odroid, they were quite expensive. The Raspberry Pi is not powerful compared to them, but because of the fantastic cheapness, it is trendy in the market. The Raspberry Pi happened cheaper than others, and, as a result, the board was not efficient enough for some tasks compared to its competitors. In particular, it is poorly suited for network tasks and USB functionality. Here is the SMSC LAN9514 chip, which connects to the SoC with one USB channel, acting as a USB-to-Ethernet adapter and USB hub at the same time. Thus, Ethernet and USB are located on the same channel and compete with each other, which contradicts the typical use of the NAS when something is downloaded over the network and stored on a USB drive, not to mention the addition of RAID here.

For the same reason, even when a model with Gigabit Ethernet support was released last year, real network performance never even came close to gigabit but was a maximum of 40 MB/s in net speed and a maximum of 20 MB/s. At that time, there were already cheap boards with real Gigabit Ethernet and USB3. A similar problem is with the Raspberry Pi, where the CPU and GPU are integrated into the same BCM2837B0 chip. The central processor is a 64-bit quad-core ARM A53 at 1400 MHz (in the Pi 3B), and the graphic processor is a dual-core 32-bit VideoCore IV with a frequency of 400 MHz [5]. The integration of CPU and GPU is widespread in the mobile world because it reduces the price and power consumption. Competi-

Table 1.2: Comparison of the characteristics of single-board computers

Title	**RAM, MB**	**CPU, GHz**	**Number of cores**	**Ethernet**	**Micro SD**	**Size, mm**	**Price, eur.**
Colibri	512	1,0	4	Yes	Yes	74 x 74	58
Orange	256	0,8	2	Yes	Yes	69 x 48	18
Raspberi	1024	1,1	4	Yes	Yes	85 x 56	65
Rock 4	2048	1,5	8	Yes	Yes	85 x 54	105

tors NXP iMX and Allwinner use a similar approach.

Thus, in the last Pi, there are six cores, but only four of them are ARM. The processor runs Linux, but GPU cores run under the real-time operating system ThreadX. This closed-source operating system manages the system without the knowledge of the Linux kernel.

In the fall of 2018, the Rx3399 board from Radxa, called Rock Pi 4, was launched. RockPi 4 is the same size as the Raspberry Pi. The board can be installed in the Raspberry Pi case. It fits but needs to be changed a bit to fit the heatsink since the processor is located on the back of the board. As far as we know, the company has improved several things compared to the Raspberry Pi, namely, power supply, the ability to use eMMC instead of an SD card, and in the future M.2 for storage. Some users, when using Raspberry Pi in projects, already had problems with power supply and SD-card. Rock Pi is still more reliable than the Raspberry Pi. The board also supports the Raspberry Pi camera and display.

Below are the comparative characteristics of popular single-board platforms (Tables 1.1, 1.2):

For our project, first of all, we need an open-source system, with open drivers and the ability to change components of both the board itself and the drivers used. Open source should fully allow the modification of any components and their use for industrial purposes. Regular support and updates from the manufacturer, industrial reliability are a critical component of the choice of equipment used. In the long run, it is essential to support the equipment used and the ability to replace with newer versions of devices without breaking backward compatibility.

Since the developed device will be used for specific goals and objectives, it is essential that it has a small size, and there are no unnecessary additions that will not be used. Based on the tables, the favorites are the Colibri and Orange Pi series devices. Therefore, we will compare them in more detail. In essence, they are approximately similar in their parameters, Orange Pi is also much cheaper than its competitor, but in the long term orange pi does not have such long-term support as Colibri until the 2032 year [10]. Another significant point in using Colibri is its reliability, which is achieved using SO-DIMM DDR3 RAM, which does not lose power and reliability due to reduced energy costs.

Based on the comparison, it was decided to pay attention to the Colibri iMX6 in terms of price and quality. Detailed description of Colibri iMX6 in Appendix A.

Colibri Evaluation Carrier Board is a debugging board for developing and debugging applications on production modules of the Colibri family based on XScale PXAxxx processors. The board has a built-in analog VGA-interface for connecting electron beam or LCD monitors, as well as a built-in connector for connecting a TFT display. Using the standard PS / 2 or USB connectors, that can connect a keyboard and mouse to the board, and audio devices using the “Jack” connectors. Also, the user interface of the developed system can contain switches, buttons, and LEDs installed on the board. Distinctive features:

- Connector for installing the Colibri XScale PXA270 module (320).
- Ethernet port 100 Mbps.
- Interface memory cards CompactFlash / PCMCIA / SDCard / Memory Stick.
- VGA interface based on ADV7125.
- two PS / 2 connectors (keyboard and mouse).
- Audio inputs (line and microphone) and line output.
- Infrared port.
- Two user buttons.
- Custom switches and LEDs.
- Channels of analog input-output.
- USB interface (two hosts and one device).
- Two serial interfaces RS-232.
- Isolated (NTE0505M) CAN interface.
- LCD backlight connection connector.
- All signals are available on connectors with a combination of inclusion.
- Power supply $+ 7 \div 24$ V.
- Power switch and a reset button.
- Debugging JTAG interface.
- Pre-installed secondary power source.

Colibri iMX6 is compatible with:

- TFT module LG.Philips — LCD TFT 6.4 LB064V02-TD01.
- Data cable (on the one hand a flat cable connected to the display, on the other hand, a two-row IDC block with a step of 2 mm) - ZU-05-319.
- DC / DC inverter — INVERTER GH025A.
- Cable for connecting to the inverter — KA-02-048.
- Touch panel - AST-065B080A.
- Connector for the Colibri SO-DIMM 200 module.

Thus, having considered all possible analogs, it was decided to use the Colibri iMX6 board. Such a choice was made based on its cost and capabilities. Colibri iMX6 can be used in complete applications for implementation on the Colibri XScale core, portable battery-powered equipment, Ethernet-based systems, video surveillance and control systems, and USB 2.0-based applications.

2 Task Planning in Operating System

2.1 General Information about Embedded Operating Systems

In the process of developing devices with a graphical human-machine interface, sooner or later, the task arises not only to create the interface itself but also to choose the platform on which it should work. Such a platform can be either a single-board computer with an operating system, or a microcontroller with a screen and a set of appropriate libraries, or some other original solution. As for the operating system for a single-board computer, let us look at two options at once, one of which is based on Windows Embedded, and the other on Linux Embedded, and compare them.

Microsoft initially introduced Windows Embedded for embedded systems. Even in the early version of Windows Embedded XP, developers were provided with a set of tools for building images that were maximally trimmed but provided with the correct drivers. These assemblies required a minimum amount of RAM and worked well even on weak processors. The technology was developed in the version of Windows Embedded Standard 7, where the process of creating your assemblies was brought to mind. Windows Embedded used the ideology of an open catalog of modules, and anyone could equip their system with only the basic set of components. The latest version in which this approach was available, albeit in a truncated form, is Windows Embedded Standard 8.1.

Modern Windows 10 IoT is positioned as an alternative to Windows Embedded but is very different from its predecessors. An open component catalog is no longer available. It is no longer possible to collect images from the designer and create your installation disks from them. Individual parameters for the embedded system, such as a write filter, bootloader branding, keyboard filter, are now configured in the pre-installed system. Besides, support for these features is available only in the latest versions of Windows 10 Enterprise. However, despite the high system requirements, especially in terms of RAM, the use of even heavy versions in embedded solutions does not cause any problems, primarily due to the availability of hardware components. Also, do not forget that in recent years it has become increasingly difficult to buy a license for old products from Microsoft.

There is no single standard way to create embedded systems with a graphical interface based on Linux. Just like there is no single typical distribution, it is still a family of systems created on entirely different principles. The Yocto Linux project may seem the closest in terms of the subject matter. However, compiling images rich in applications, drivers, and third-party libraries is quite tricky. It is easier to take a ready-made, well-supported distribution and configure it manually to solve this problem.

Since the system is based on a single-board computer, it is easy to install any touch screen or other controls. It is possible to create not only the device management console or the interactive dashboard but also other devices, such as information or trading terminals.

There is an opinion that Linux is complicated to configure and maintain, but it is free, and Windows is simple and convenient, but it costs money. Perhaps this is so, but only at the household level. When it comes to creating embedded systems, there are other circumstances to consider. For example, fine-tuning windows, especially in terms of power management and other low-level elements, no longer seem easy and convenient. Besides, do not forget that Windows 10 quite resource intensive. In this example, we were able to measure the memory consumption of a clean system several times, and it was about 400 MB in the standby state. In comparison, Linux Debian, with all its extensions, used about 200 MB of memory. Of course, if there are several gigabytes of RAM, this is not a problem, but still, when using Windows and highly loaded client applications, that will have to take more powerful single-board computers. Linux is less demanding on resources, but it is complicated to configure and requires a thorough approach, especially when working with the bootloader. Moreover, in the development and implementation of some aspects of the system, sometimes it becomes necessary to build an individual kernel with unique parameters. It is effective but requires an appropriate level of skill.

It is difficult to make a clear choice when creating a secure interface. The complexity of deploying systems is about the same. The cost of acquiring Windows licenses for several devices can be compared with the cost of maintaining systems on Linux. The Linux operating system has long established itself as a reliable and effective means of solving any problems. In many respects, it has no equal in the field of server applications. Some Linux distributions are so adapted for the work of an ordinary user and solving his daily tasks that they can safely bear the name of a desktop working system, slightly inferior to Microsoft Windows [7].

Thus, the secret of Linux success lies in its non-profit nature and open source. Any developer with the desire and necessary knowledge can make a change to the Linux code, starting with graphical applications and ending with the kernel of the operating system. Not a narrow circle of programmers is working on Linux, but the global community of developers. It is for this reason that Linux is developing rapidly, continually updating the components of the operating system. If an error is detected in the code, it is corrected in the shortest possible time. The foundation of the Linux operating system is its kernel, which, like all other components, is periodically updated. Kernel subsystems modify to optimize its operation, adding support for new hardware.

2.2 Overview of Existing Linux Task Schedulers

2.2.1 JobScheduler

One of the popular tools for automating work in them is JobScheduler, but for beginners, this scheduler seems too complicated. When talking about the task scheduler, Linux often means Cron. Nevertheless, to perform specific tasks on a schedule, or periodically, after an equal period (regular actions), it is better to use JobScheduler. This is a daemon program (a service, by analogy with Windows), which is constantly located and works in the computer's RAM, scanning once a minute the list of installed tasks for the actual execution for the current time [7]. The instructions for starting tasks themselves are in specially designed tables, which, in turn, are located in strictly reserved system directories. It consists of 6 columns, separated by tabs or spaces, the first 5 of which determine the start time of the action. The last column is interpreted as a start command, i.e., the action itself. JobScheduler files cannot be edited manually, because this may cause a failure in the processing cycle of these files when the daemon scans them to analyze the description of actions. There are main features:

- Job activities can be limited to duration, supports any number of time slots.
- The JobScheduler uses job priorities for managing tasks.
- JobScheduler stores job history and logs.
- Only one process can access the resource with lock it for others.
- Main job packages for log rotation and cleanup, sanity checking, job execution by remote.
- Notifications with job state and results with logs by e-mail.
- API for creating and running jobs. supporting graphical user interface to better controlling tasks.

2.2.2 Activebatch

Advanced Systems Concepts' distributed management and scheduling system ActiveBatch Version 3.0 provides batch jobs on heterogeneous platforms, including Windows NT, UNIX, and OpenVMS. ActiveBatch 3.0 is a multi-functional and easy-to-use system. The graphical user interface offers various views for both beginners and specialists, and program experts of queues and tasks simplify the planning and execution of tasks. Independent of the scripting language, it can also be used to submit tasks using languages such as XLNT, Perl, VBScript, and Java [5]. The ActiveBatch scheduling system allows completing tasks depending on time or event. The product also has a Runbook View for recording and analyzing all batch jobs — including scheduled, ongoing, and completed ones. The presence of task logging functions ensures the audit of all current information about them. The system can monitor tasks for failures and allows them to set warnings/interrupts. The task restart function makes it possible to resume work in the event of a system crash automatically. ActiveBatch provides automation operators with access to any access point and around-the-clock management, so they can safely control and manage critical operational and business processes while away from their workstations. Custom alerts and dashboards provide real-time display of the automation environment to notify users of business-critical issues, facilitate quick responses, and effectively troubleshoot problems [7]. Major features for increased flexibility for ActiveBatch operators:

- easily view, control and manage ActiveBatch alerts.
- receive push notifications to raise awareness and take action on significant conditions.
- monitoring of active instances, historical data, and logs.
- trigger, restart, enable and disable key work processes from anywhere at any time.
- viewing information about templates, instances, and audits.
- create, configure and filter the application toolbar.
- secure user management with granular political and group permissions and management settings (installed in the ActiveBatch console).

2.2.3 Oracle Grid

Oracle offers a new Scheduler tool that helps automate tasks inside the database. The DBMSSCHEDULER package contains various functions and procedures that help manage the scheduler, although it is also elementary to schedule tasks to run in the Database Control interface. The most important architectural feature of the scheduler is its modular approach to task management, which allows reusing similar tasks [11]. Using scheduler together with Database Resource Manager allows for fine-tuning the allocation of resources to various tasks. The Oracle Scheduler is not only a job description tool, it also helps control resource utilization and prioritizes tasks within the database. Scheduler consists of five fundamental components — jobs, schedules, programs, events, and chains. Tasks are very similar to the tasks in the main package, but schedules, programs, events, and chains are new concepts that form a modular approach to managing tasks. A program, for example, allows multiple users to perform similar tasks.

A job is a planned task for single or multiple automatic starts. The assignment contains a specification of what should be done and when. A Scheduler job can execute a PL/SQL code block, a native binary executable, a Java application, or a shell script. A new job is created by specifying its details, such as the actions to be performed, the time and frequency of the launch — as is done using the traditional package. The scheduler allows abstract all the details of the execution and starts a time of a task using program modules or schedules.

A schedule is a specification of when and how often a database should complete a task. The same schedule can be used for multiple jobs. The schedule can run tasks when a specific event occurs in the database [9].

The program contains metadata about the job. The program includes the name, type (for example, PL / SQL code or a UNIX shell script), and the action of the program, which is the actual name of the procedure or executable script. Please note that the task can specify what exactly should be done directly in the definition of the task, or use a previously created program for the same purpose [9].

The scheduler uses the Oracle Streams tool to trigger events and run event-based database jobs. An event is a message sent by an application or process in response to an action or condition. There are two types of events — events triggered by a Scheduler, and events triggered by an application. Scheduler triggered events are triggered by changes in the functioning of the scheduler, successful completion of a job can also be an event. Application-triggered events are consumed or used by the scheduler to start the job. As a means of launching a task, it is possible to use an event instead of a job. In this case, the schedule is called the event schedule.

The concept of the Scheduler chain is used to link together interdependent programs. Thus, the launch of a specific program may be linked to the successful execution of some other programs. It is possible to run a chain-based task instead of the only scheduled program. When there are interdependent tasks, the chain makes it easy to run all the programs needed to complete the whole transaction.

One of the limitations is that it can only perform PL / SQL — based tasks, and it cannot be used to schedule the launch of system scripts or executable files. To run such non-database jobs, have to use the crontab tool on UNIX or at on Windows servers, or use third-party tools. Oracle Scheduler allows using PL/SQL scripts, operating system shell scripts, Java programs, and native binary executables to complete scheduled tasks.

Considering the popular event planners, considering their strengths and weaknesses, it was decided to opt for JobScheduler as the most universal in terms of the type of database, supported platforms, output to file, check results of tasks, relaunch, audit, integration and remote control.

2.3 Overview of Task Scheduler on Windows 10

Having considered the possible options for task schedulers in the Linux embedded system, the project also needs the ability to equation tasks based on a regular user computer. According to the data provided on the website Market Share Statistics for Internet Technologies [12], which tracks the popularity of operating systems used by users, according to the latest data for February 2020, Windows 10 is the most popular system. It occupies a market share of more than 57% by the current moment (Table 2.1).

Table 2.1: The popularity of operating systems

Name of the system	**Share, %**
Windows 10	57.22
Windows 7	25.39
Windows 8.1	3.43
Mac OS X 10.15	3.43
Mac OS X 10.14	2.96
Mac OS X 10.13	1.53
Windows XP	1.34
Linux	1.29
Mac OS X 10.12	0.7
Others	2.71

According to Microsoft's statistics, Windows 10 is currently installed on more than 900 million devices. The company plans to reach 1 billion devices by the end of 2020. Also, in January 2020, support for the second polarity of Windows 7 was completed, which would only increase the popularity of Windows 10. The growth of its popularity is getting higher and higher every year, and this gives reason to choose it as the underlying system for users to use.

Unlike Linux, in Windows 10, its task scheduler called "Task Scheduler" is more common. In essence, it is a snap-in MMC (Microsoft Management Console), with which you can assign various tasks that will be performed at a specific time or when certain events occur. Typically, such tasks are used to automate individual processes:

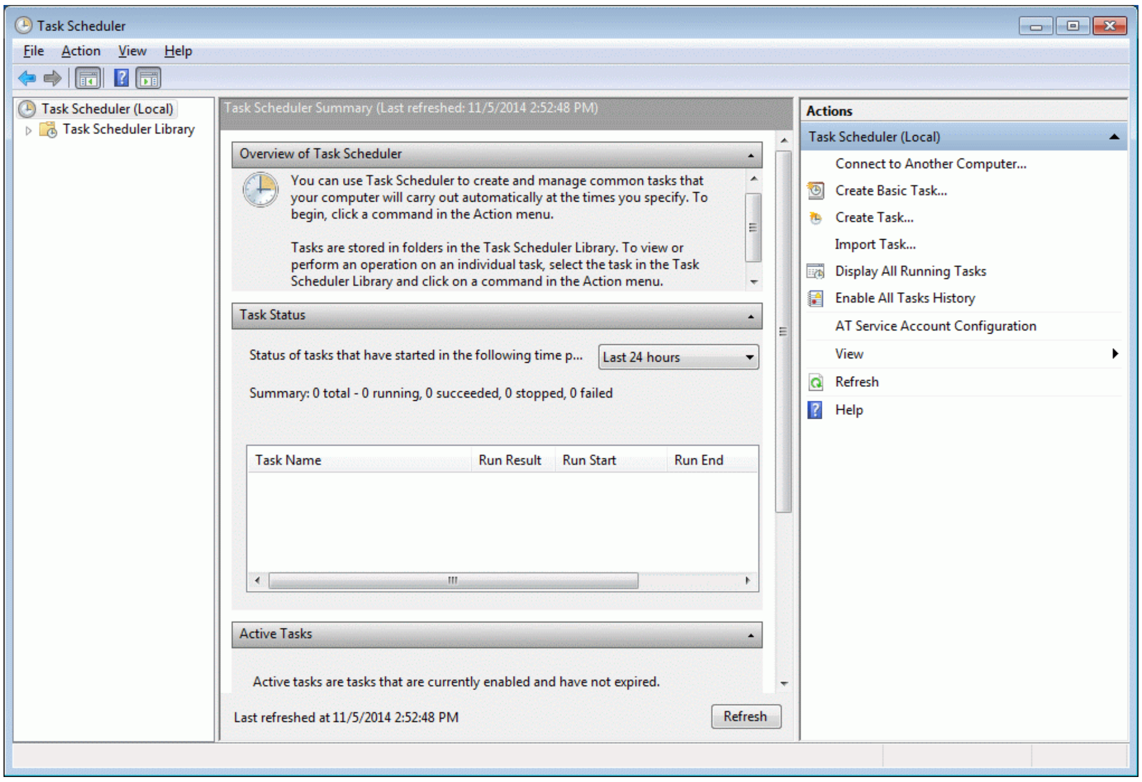
- Parametric automation of various tasks performed on a computer.
- Optimization of the computer boot process.

The Windows 10 operating system contains several task scheduling tools, including such as Task Scheduler, the Schtasks command-line tool, and several Windows PowerShell console cmdlets. These tools can be used to schedule tasks at both local and remote workstations [13].

Jobs can have various properties associated with them, including the following:

- `Triggers` – Using triggers can set the conditions for starting and completing various tasks. Tasks can be performed on schedule when a user logs on to the system when the computer starts. You can include events related to user actions in the task launch parameters. The use of event triggers significantly enhances process control capabilities.
- `Actions` – The task parameter determines the specifics of the running process. Allows a process to run programs, send email messages, or display messages.
- `Conditions` – The task parameter specifies the events in which the active process starts or stops. For example, under a given condition, you can start or

stop a task based on the length of time the computer is idle. Conditions can be used to wake the computer from sleep mode to complete a task. You can configure the conditions for the task under the condition that the computer is running on the network.


Task Scheduler

File Action View Help

Task Scheduler (Local)
Task Scheduler Library

Task Scheduler Summary (Last refreshed: 11/5/2014 2:52:48 PM)

Overview of Task Scheduler

You can use Task Scheduler to create and manage common tasks that your computer will carry out automatically at the times you specify. To begin, click a command in the Action menu.

Tasks are stored in folders in the Task Scheduler Library. To view or perform an operation on an individual task, select the task in the Task Scheduler Library and click on a command in the Action menu.

Task Status

Status of tasks that have started in the following time p... Last 24 hours

Summary: 0 total - 0 running, 0 succeeded, 0 stopped, 0 failed

Task Name	Run Result	Run Start	Run End

Active Tasks

Active tasks are tasks that are currently enabled and have not expired.

Last refreshed at 11/5/2014 2:52:48 PM Refresh

Actions

Task Scheduler (Local)
- Connect to Another Computer...
- Create Basic Task...
- Create Task...
- Import Task...
- Display All Running Tasks
- Enable All Tasks History
- AT Service Account Configuration
- View
- Refresh
- Help

Figure 2.1: The main window of the Task Scheduler on Windows 10

To work with a task can right-click on it in the main panel and select one of the following commands in the context menu:

- `Run` – run the task for execution;
- `Finish` – if the task is running, stop its execution.
- `Disable` – temporarily disable the task;
- `Export` – export the task to a file that can be imported to another computer;
- `Properties` – view and edit task properties;
- `Delete` – completely delete the task;

3 Task Manager API

3.1 JobScheduler API

Instructions for starting tasks are in specially designed tables, which, in turn, are in strictly reserved system directories. They consist of 6 columns, separated by tabs or spaces, the first 5 of which determine the start time of the action. The last column is interpreted as a start command, i.e., the action itself. JobScheduler files cannot be edited manually, because this may cause a failure in the processing cycle of these files when the daemon scans them to analyze the description of actions.

JobScheduler is an open-source solution for enterprise process automation. JobScheduler is used to run executable files and shell scripts, as well as to run database procedures automatically. JobScheduler stores all the information in the server database system.

Its primary function is to run executable files, shell scripts, and database procedures. Used for batch scheduling, the JobScheduler wizard runs as a Unix daemon or Windows service in the background. Jobs are run on the primary instance of JobScheduler or any universal JobScheduler agent.

Job management is done using the command line or the built-in graphical user interface JOC (JobScheduler Operations Center). To configure the job, the XML JOE editor (Job Object Editor) is available.

3.1.1 Additional JobScheduler Features

High Availability Cluster: JobScheduler redundant cluster provides uptime and fault tolerance with automatic failure. A fault-tolerant system consists of a primary JobScheduler and at least one backup, both of which JobScheduler works on different computers.

Load balancing: for large amounts of data with long processing times using multiple JobSchedulers will significantly speed up processing time and provide higher availability. In load-sharing mode, processing tasks are distributed among several JobSchedulers that process distributed orders on more than one host.

JobScheduler Operations Center (JOC) Cockpit is the end-user interface for JobScheduler. Designed to keep things simple and minimal, JOC Cockpit offers a modern, responsive user interface, web service APIs, and finely tuned authentication and authorization for scheduling open source jobs. The JOC cockpit completely replaces and extends the functions of the classic JOC and the JobScheduler, JID dashboard,

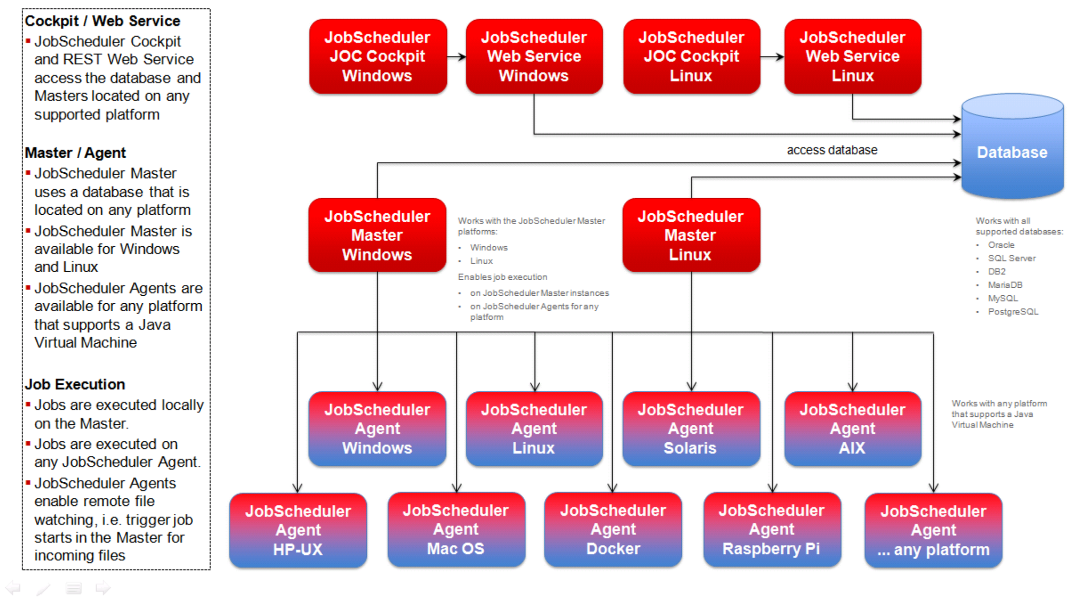

Cockpit / Web Service
JobScheduler Cockpit and REST Web Service access the database and Masters located on any supported platform
Master / Agent
JobScheduler Master uses a database that is located on any platform
JobScheduler Master is available for Windows and Linux
JobScheduler Agents are available for any platform that supports a Java Virtual Machine
Job Execution
Jobs are executed locally on the Master.
Jobs are executed on any JobScheduler Agent.
JobScheduler Agents enable remote file watching, i.e. trigger job starts in the Master for incoming files
JobScheduler JOC Cockpit Windows
JobScheduler Web Service Windows
JobScheduler JOC Cockpit Linux
JobScheduler Web Service Linux
Database
access database
JobScheduler Master Windows
Works with the JobScheduler Master platforms:
Windows
Linux
Enables job execution
on JobScheduler Master instances
on JobScheduler Agents for any platform
JobScheduler Master Linux
Works with all supported databases:
Oracle
SQL Server
DB2
MariaDB
MySQL
PostgreSQL
JobScheduler Agent Windows
JobScheduler Agent Linux
JobScheduler Agent Solaris
JobScheduler Agent AIX
Works with any platform that supports a Java Virtual Machine
JobScheduler Agent HP-UX
JobScheduler Agent Mac OS
JobScheduler Agent Docker
JobScheduler Agent Raspberry Pi
JobScheduler Agent ... any platform

Figure 3.1: Architecture of JobScheduler [14]

so one interface is enough to perform all tasks — from monitoring tasks and task chains to checking history logs. The JOC cockpit is installed separately from the JobScheduler wizard. This separation of functions provides the best user experience for service operators and allows the use of JobScheduler in security-sensitive and critical networks: JOC can be operated in less restrictive network areas than the Master and JobScheduler agents.

JobScheduler REST Web Service is a replacement for the obsolete XML interface. The REST web service is used by the JOC cockpit and external applications to interact with the JobScheduler wizard. The REST web service provides access to such objects as tasks, task chains, and orders, as well as their corresponding operations, for example, to add an order to the task chain (Figure 3.2).

Table 1 in Appendix C lists the available XML commands for the JobScheduler REST API web service.

In this way, JobScheduler comes with a powerful REST web service interface for accessing external applications. This interface is used for tasks such as adding orders, adding events, and receiving information about the status of tasks and orders.

3.2 Windows Task Scheduler

The Task Scheduler is util of Microsoft Windows allows predefined actions to the launch of programs or scripts whenever a specific set of conditions or after specified time intervals. For example, you can schedule a task to run a backup script every night, or send you an e-mail whenever an individual system event occurs.

Tasks can be scheduled to execute in response to these events, or triggers [13].

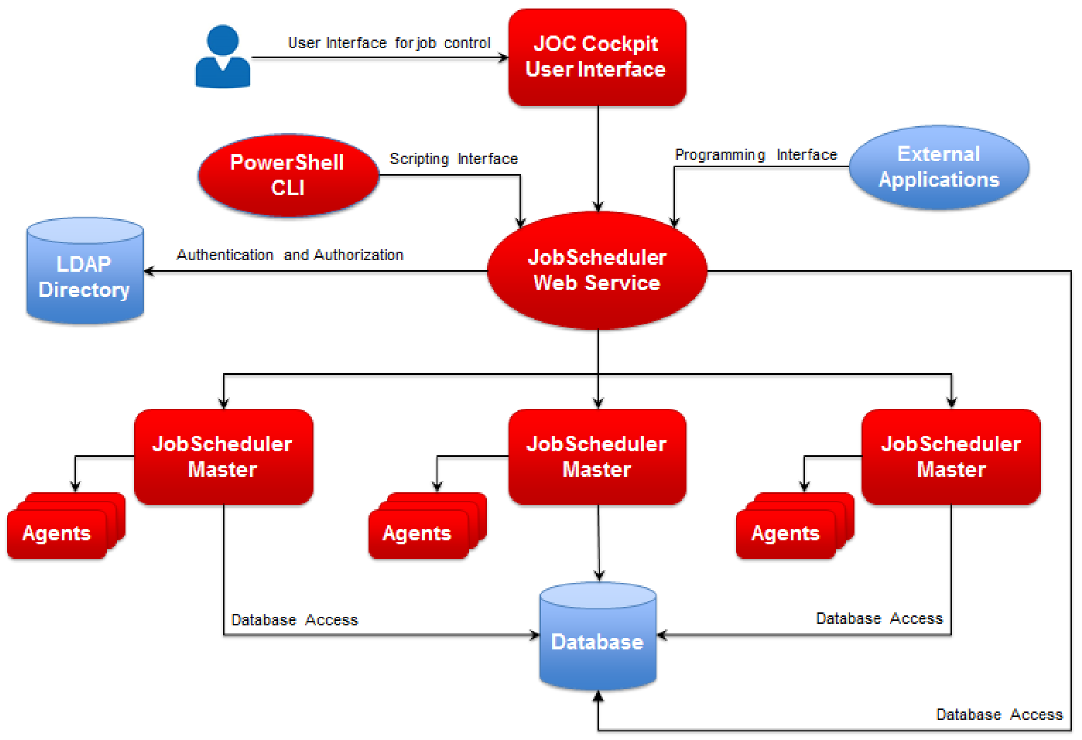


Figure 3.2: JobScheduler's Web Services [14]

- When a specific system event occurs.
- At a specific time.
- At a specific time on a daily schedule.
- At a specific time on a weekly schedule.
- At a specific time on a monthly schedule.
- At a specific time on a monthly day-of-week schedule.
- When the computer enters an idle state.
- When the task is registered.
- When the system is booted.
- When a user logs on.
- When a Terminal Server session changes state.

Consider how to create scheduler jobs using the API in Appendix D.

4 JobScheduler

4.1 Preparation

JobScheduler requires a Java virtual machine for its operation. Before starting the deployment process, make sure that the Java Runtime Environment (JRE) is installed on the machine on which the scheduler will run.

Besides, we need to prepare the database server. It can be launched both locally and remotely.

This software is available both for operating systems of the Linux family and for operating systems of the Windows family. The installation procedure is identical in both cases and described in Appendix B.

4.2 Job and patterns

A job determines the program to be executed, its run time and what is to be done in the event of an error occurring. Further, any parameters to be used, pre and post processing, locks preventing simultaneous access to a file and possible follow-on jobs may also belong to a job configuration. The XML configuration of a job can be carried out in the central start configuration file, (which is usually ./config/scheduler.xml) or in a separate configuration file in the configuration directory which is monitored by the JobScheduler.

Jobs are the basic unit for processing executable files, shell scripts, database procedures, and job implementations based on the JobScheduler API. Tasks can be performed independently of each other. Depending on the result of the execution, i.e., the exit code that signals success, failure, or a specific exit code, the task can run any number of subsequent tasks. Jobs can run in parallel up to a configurable number of simultaneous tasks. Chains of tasks can be considered as an assembly line along which several nodes of tasks are transmitted. Thus, each task contains precisely one step in processing the chain of tasks. Job dependencies based on the results of the respective job nodes can be configured for the job chain.

Directory monitoring can trigger jobs to run, allowing them to integrate legacy applications into the business process.

Jobs perform file transfers based on the built-in capabilities of FTP and SFTP as a popular application integration tool.

Launching tasks is launched by the built-in calendar, command line, or web interface. Other applications may run jobs or otherwise control the JobScheduler using the API.

Work may be limited to time intervals. JobScheduler supports any number of time slots that can be customized to suit individual requirements.

JobScheduler allows prioritizing work. Job history logs are optionally stored in a database. The lock function prevents two jobs from simultaneously accessing the same resource, such as a file or database. In other words, only one process at a time can get the exclusive right to access the resource while the lock is active. Standard job packages, for example, for rotating logs and cleaning, checking sanity, performing tasks with remote JobSchedulers, and transferring files. Email notifications of job results, customizable logging, and log monitoring. An API for implementing tasks and job scripts, for example, for complex conditional processing. There are various graphical user interfaces: an integrated interface for job management and a graphical interface for configuration management for several JobSchedulers on different server systems.

There are some job patterns that can use. Figure 4.1 shows ”Independent Job Start Pattern”. Jobs are completely independent from one another. Each job has an individual start time.

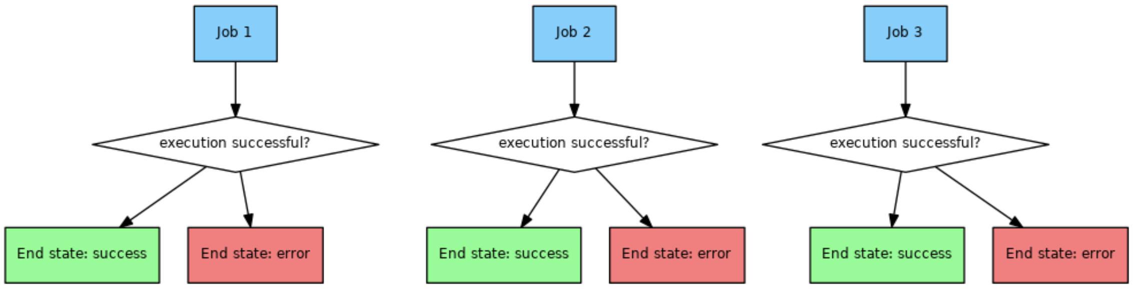


Figure 4.1: Independent Job Start Pattern [14]

The another pattern when job starts based on execution results. Jobs are started based on the execution result of a previous job. Figure 4.2 shows if Job 1 returns an exit code 1 then Job 2 will be executed. If Job 1 returns an exit code 2 then Job 3 will be executed.

The one more popular pattern that use in current project is ”Parallel Tasks Job Start Pattern”. Figure 4.3 shows that jobs can be executed in multiple parallel tasks. All tasks are executed independently from one another.

Jobs have one of the following statuses:

- `Pending` – No task is running. This is the starting status.
- `Running` – At least one task is running.
- `Stopping` – The job is stopping. The JobScheduler will not start another task and all current tasks are being stopped. As soon as all tasks are stopped, the job status changes to stopped.
- `Stopped` – No tasks are running and no further tasks will be started by the JobScheduler.

- Read_error – The reread command has caused an error and the job is unusable as the program code cannot be read from the underlying file.
- Error – The JobScheduler does not start any new tasks after an error has occurred.

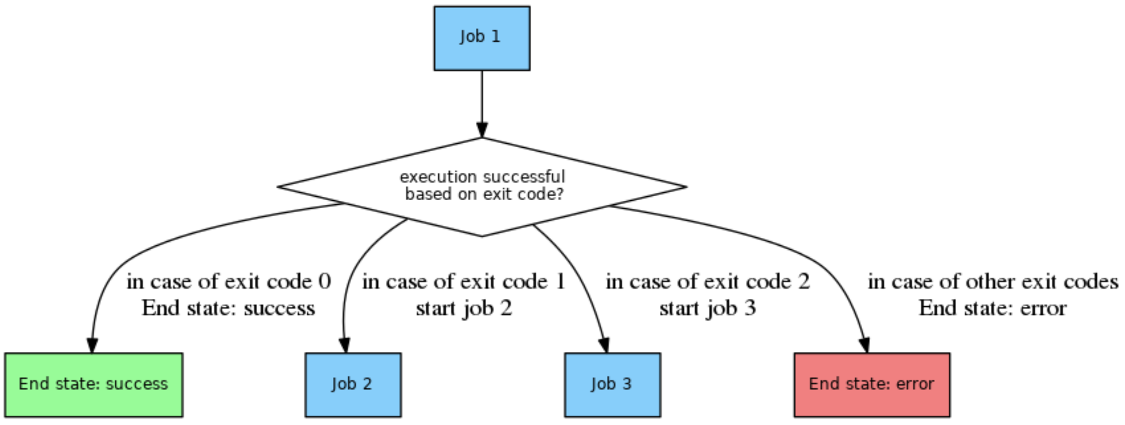

Job 1
execution successful based on exit code?
in case of exit code 0 End state: success
in case of exit code 1 start job 2
in case of exit code 2 start job 3
in case of other exit codes End state: error
End state: success
Job 2
Job 3
End state: error

Figure 4.2: Pattern based on Execution Results [14]

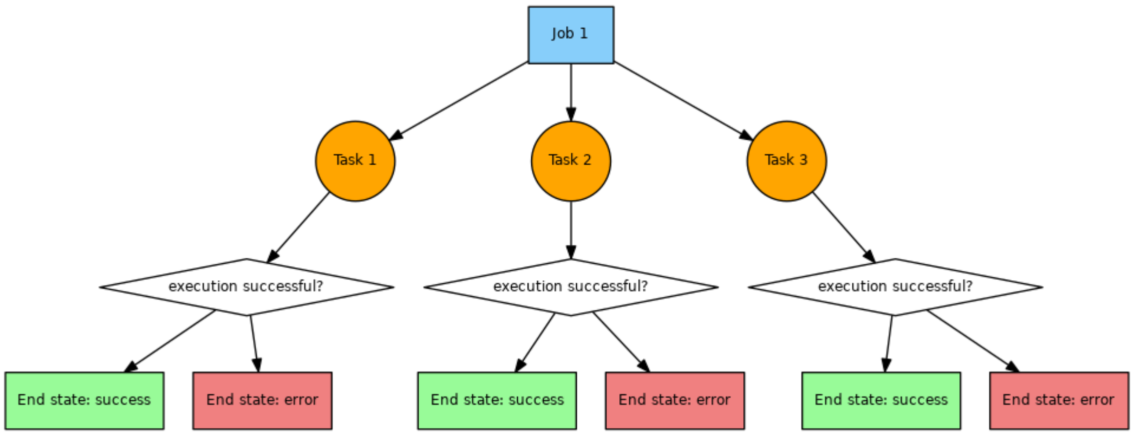


Figure 4.3: Parallel Tasks [14]

4.3 Job execution

The <run_time> parameter is used in the configuration to specify whether a task should be started once or repeatedly. Both the <start_job> and the Job.start() API method can be used to start a task.

A task will start automatically when no other task is running and when one of the following conditions are true:

- At the start of a <period>, when repeat= or single_start= is specified in the period.
- When a previous run set Task.repeat and the repeat time has been reached.
- When a previous run caused an error and Job.delay_after_error has ended.
- When the interval after the end of the previous task defined in <period repeat="..."> has ended.
- When directory monitoring is active and a change occurs in the monitored directory.

In addition, a task will start when: An order for the job is present and the number of tasks running is less than that specified in <job tasks="...">. A task will only start when it has a start time (at) or:

- the job has not been stopped,
- a period for the current time is given,
- the (Job.delay_after_error) delay after an error is not active.

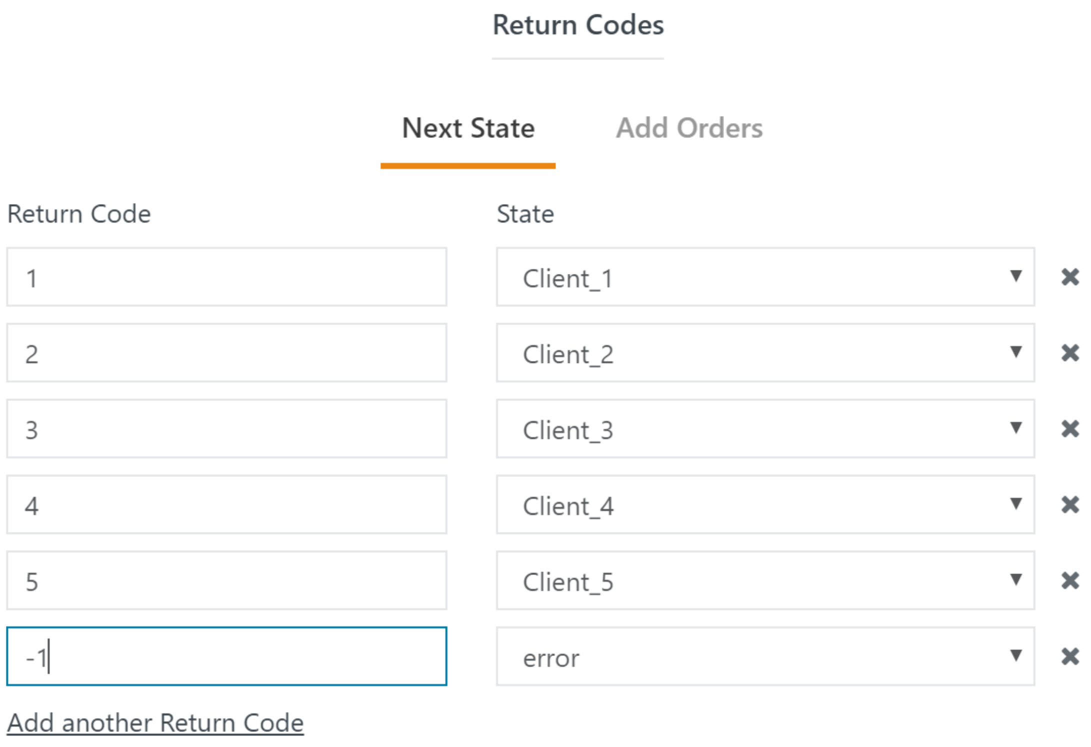
After task of a job has terminated you can call various commands depending on its exit code. These commands are particularly "start_job" and "add_order".
Both commands are supported in JOE:

- Go to command
- Add a new command
- Set the exit code for which the command will be called
- Use the add job or add order buttons to select the command

There are other rare use cases for commands (see list of XML commands).
Other commands may also be indicated in JOE, however, you have to use the "edit XML" function in the context menu of a job to add such commands. A panel will be displayed where you can insert the command.
The exit code of a PowerShell/Batch script is usually expected as the result of the execution of the script. However, the program returns the exit code of the execution of itself - and this in most cases 0.
To retrieve the effective exit code of the script, that script has to be terminated with an exit() function and as parameter the variable $lastexitcode (or any other value/variable for the exit code).
The example below shows how it works:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job title="How to get the exit code from a powershell script"
    name="PowerShellExitCode">
   <script language="shell">
       <![CDATA[
powershell.exe -noprofile -command "write-output test; exit 123 "
echo %errorlevel%
exit %errorlevel%
       ]]>
    </script>
    <run_time/>
</job>
```

There is an example of how to configure a job chain with return codes. There is a one start job, it has some logic after that should run different jobs. Open job return codes and configure it (Figure 4.4). When the job finished with the return value is 1, then Client_1 will be started, and so on. In the case of error job emit the return value ”-1” and error job will be start.



Return Code	State
1	Client_1
2	Client_2
3	Client_3
4	Client_4
5	Client_5
-1	error

Figure 4.4: Job’s return codes

Figure 4.5 shows how seems the infrastructure of jobs chain with return code.

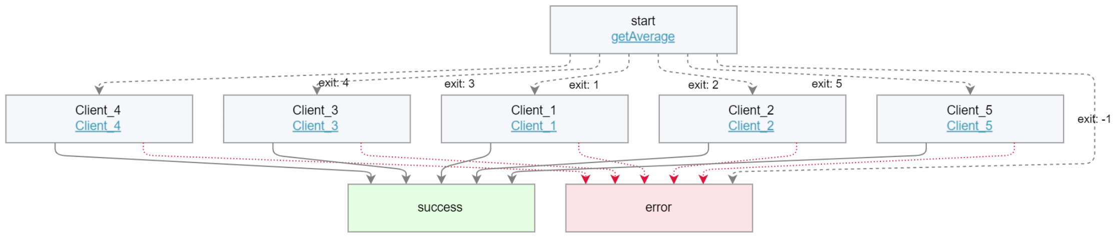


Figure 4.5: Jobs chain with return codes

4.4 JobScheduler requests

After all the necessary software has been installed, and all the services have been started, we should make sure that everything works.

JOC Cockpit provides a web interface that allows us to monitor the status of the JobScheduler, as well as perform all the manipulation of tasks through it.

To check if the JOC is working, we will use a browser and go to the IP address of the host where the JoSccheduler infrastructure is deployed. In this case, we log in from the same machine, so the address is as follows: http://localhost:4446.

Here, 4446 is the port number on which the Jetty webserver runs. Use the port that was specified in step 4 when installing JOC in the Jetty configuration step.

If the browser successfully displays an authorization window, this means that the JOC and Jetty services have been successfully installed and started.

Initially, a user with the root name and password root is registered on the system. It has all the rights and is intended for the initial setup of the infrastructure. Log in using the root account.

A panel of the web interface displays a map of the configured JobScheduler infrastructure. In this example, the figure shows that one instance of JobScheduler is running, and one database server is being used.

If the map shows that both the database and JobScheduler are working, then the installation and configuration were performed correctly.

Next, make sure that the REST API is operational. To do this, we need an application through which we can send arbitrary HTTP-requests, use arbitrary request headers, put any data in the request frame.

In this case, we will use Postman software, which is distributed freely and is available on the developer's website.

Before sending any commands to the REST API, the client application must log in to receive an access token. To do this, send a request with the POST method to the address /joc/api/security/login. The request should contain the Authorization header, the contents of which are written in the following format: the word "Basic", followed by space and the string "login: password" (a string in which a colon separates the login and password), encoded using the Base 64 algorithm.

In this case, we want to log in as root. The "root:root" line encoded by the Base 64 algorithm is as follows: "cm9vdDpyb290".

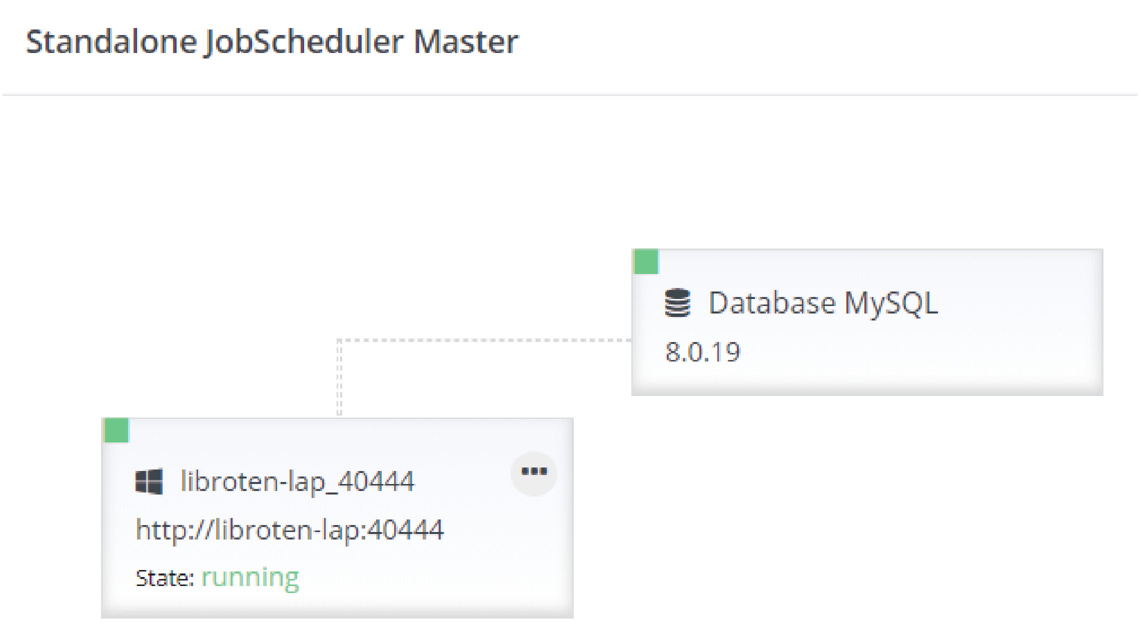
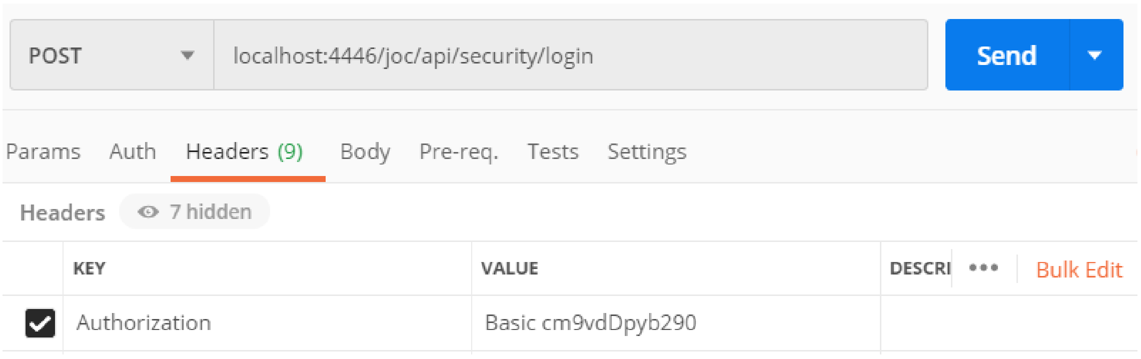


Figure 4.6: Infrastructure map

Thus, the following request must be sent to the server:

```
POST /joc/api/security/login HTTP/1.1
Host: localhost: 4446
Authorization: Basic cm9vdDpyb290
Cache-control: no-cache
```



KEY	VALUE	DESCRI
✓ Authorization	Basic cm9vdDpyb290	

Figure 4.7: Submitting a REST API request from a Postman application

The response from the server comes in JSON format and looks like this:

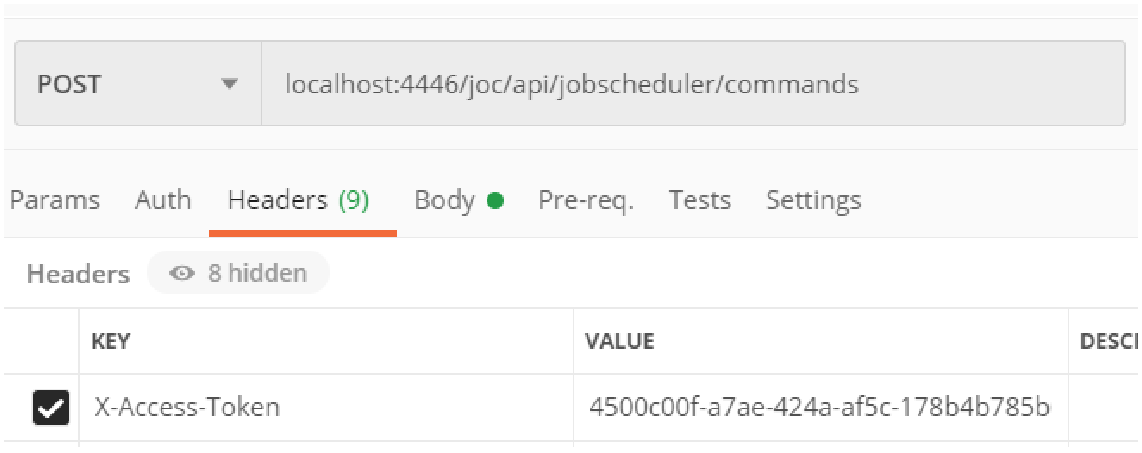
```
{
    "enableTouch": true,
    "sessionTimeout": 900000 ,
    "user": "root",
        "accessToken": "4500c00f-a7ae-424a-af5c-178b4b785b6b",
        "callerHostName": "0: 0: 0: 0: 0: 0: 0: 1",
        "callerIpAddress": "0: 0: 0: 0: 0: 0: 0: 1",
        "hasRole": false,
        "isAuthenticated": true,
        "isPermitted": false
    }
```

As a result of authorization, the following token was received from the server: “4500c00f-a7ae-424a-af5c-178b4b785b6b”.

For subsequent calls to the JobScheduler via the REST API, should contact the address /joc/api/jobscheduler/commands and transfer the received token in the X-Access-Token header.

Let us try to get information about the status of JobScheduler on this node. To do this, send the following POST request:

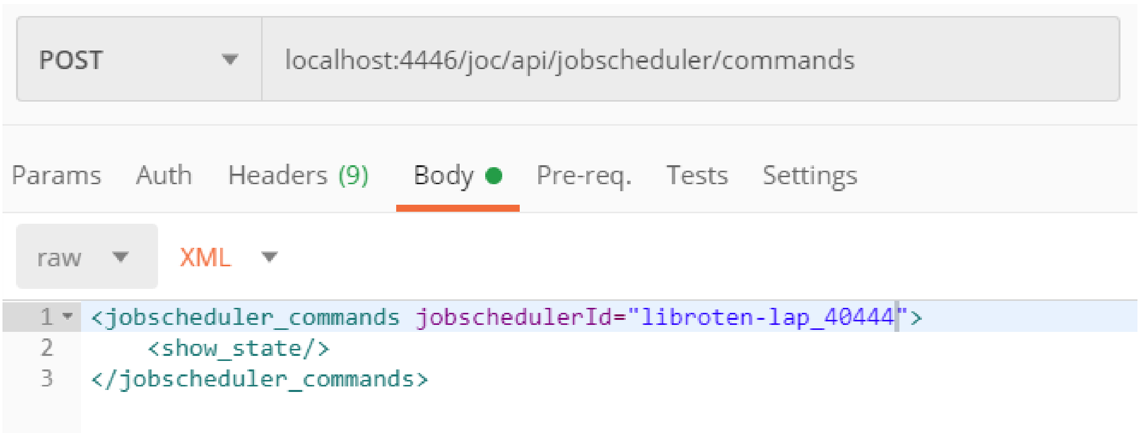
```
POST /joc/api/jobscheduler/commands HTTP/ 1.1
Host: localhost:4446
X-Access-Token: 4500c00f-a7ae-424a-af5c-178b4b785b6b
Content-Type: application/xml

<jobscheduler_commands jobschedulerId = "libroten-lap_40444">
              <show_state />
</jobscheduler_commands>
```


POST localhost:4446/joc/api/jobscheduler/commands

	KEY	VALUE	DESCI
✓	X-Access-Token	4500c00f-a7ae-424a-af5c-178b4b785b	

Figure 4.8: Scheduler Status Request — Request Headers

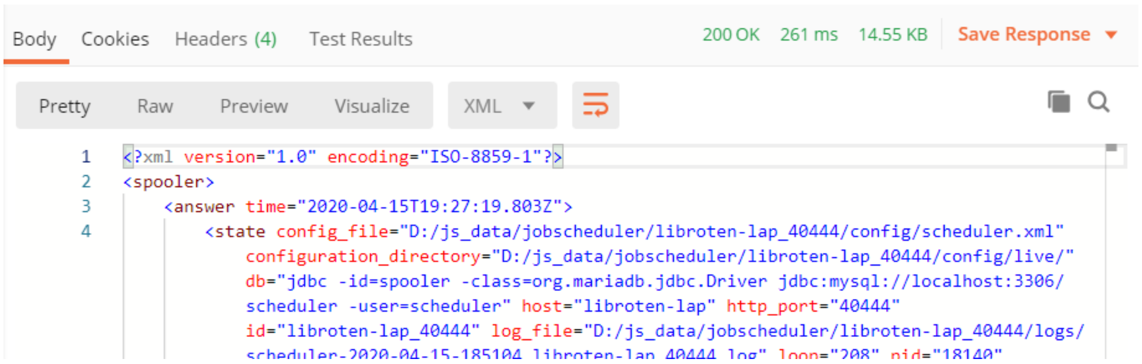
It should be noted that the JobschedulerId attribute requires the identifier of the JobScheduler process, which was set in step 2 during the installation of the scheduler.



```
localhost:4446/joc/api/jobscheduler/commands
```

```xml
<jobscheduler_commands jobschedulerId="libroten-lap_40444">
    <show_state/>
</jobscheduler_commands>
```

Figure 4.9: Scheduler Status Request — Request Body



```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<spooler>
    <answer time="2020-04-15T19:27:19.803Z">
        <state config_file="D:/js_data/jobscheduler/libroten-lap_40444/config/scheduler.xml"
            configuration_directory="D:/js_data/jobscheduler/libroten-lap_40444/config/live/"
            db="jdbc -id=spooler -class=org.mariadb.jdbc.Driver jdbc:mysql://localhost:3306/
            scheduler -user=scheduler" host="libroten-lap" http_port="40444"
            id="libroten-lap_40444" log_file="D:/js_data/jobscheduler/libroten-lap_40444/logs/
            scheduler-2020-04-15-185104_libroten-lap_40444.log" loop="208" pid="18140"
```

Figure 4.10: Scheduler Response to Status Request

4.5 Data collection and processing

To verify the server's performance on real-world tasks, create the following typical data collection task.

Remote servers host CSV files; for simplicity of the example, they contain only two lines: id and value. The CSV file is compressed using an archiver with the zip extension. Let us create several jobs in jobScheduler, in this case, 5. Each of the jobs will download an archive from its server every 10 minutes, which it will unzip and send the value as a result. In this case, the temporary folder with the downloaded files is deleted.

In this case, we can use the result code of the task, which we will pass on. In general, there is an implementation with locking values and files that can be processed using the mutex. After the successful execution of jobs, the scheduler will run the next job, which calculates the average value.

In each of the tasks, there will be a similar script

```
mkdir tmp1
cd tmp1
curl -o 1.zip https://testingtul.000webhostapp.com/1.zip
tar -xf 1.zip
Start-Sleep -Milliseconds (Get-Random -Minimum 300 -Maximum 9000)
$output = Import-Csv 1.csv | select -ExpandProperty value
Set-Content -Path '1.txt' -Value $output // for mutex realization
cd ..
rm -force -recurse tmp1
Exit $output
```

The script also contains a random delay to generate the processor with complicated and cumbersome tasks.

Task Scheduler implements event chains that help track task progress. A visualized representation of the chain of events can be seen in the figure.

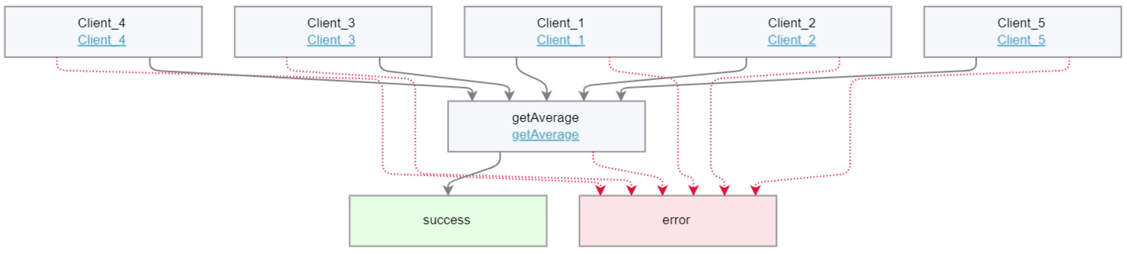


Figure 4.11: Jobs chain

In case of an error, after 10 minutes, the tasks will be restarted automatically. It is possible to configure sending messages to e-mail with an error code and additional information.

The result of the task scheduler is to obtain the necessary information from clients, which will be processed, and subsequently, the average value among all received data is displayed. The task scheduler allows you to run more than 200 similar tasks with 256MB RAM while using the task processing system in the operating system using priorities and sorting by the runtime. The solution works on Linux and Windows operating systems, successfully managing several tasks simultaneously at a maximum speed of up to 100 Mb/s.

At the moment, the work is not finished, because JobScheduler could not be launched under embedded Linux. This is due to the fact that this process is fraught with the following difficulties: JobScheduler uses such fairly heavyweight components as a Java virtual machine, an integrated webserver. JobScheduler also manages tasks using databases.

It is supposed to use the following solutions:

- An attempt to start the JobScheduler infrastructure using a version of the Java virtual machine for embedded devices.

- Deploying a database for JobScheduler on a server remote concerning the end device and setting up the scheduler to work with a remote database server.

- Install on the end devices versions of the Windows operating system for embedded devices or the Internet of things.

To complete these steps, we will need to conduct additional research and verify the applicability of these methods in practice.

5 Server for Monitoring Schedulers and Tasks on Devices

The developed server is a prototype and implements the following features:

- Storage of information about devices to be monitored.
- Connect to devices and request information about the status of the scheduler on them.
- Connecting to devices and requesting information about the tasks created on them: task name, title, enabled/disabled, status, next run time, source code of the script launched by the task, information about the language in which the script is written.
- Providing an API by which client applications can receive all of the above information about all monitored devices and their tasks.

The main goal of developing this server is to centralize the monitoring of many devices that use the task scheduler. The use of this server will allow us to carry out the monitor multiple devices through a single node without addressing each device individually.

The server is designed to work with devices that use JobScheduler software configured as a task scheduler, which is configured to remotely control via the network using the REST API provided by JOC Cockpit.

5.1 Architecture

The server consists of 4 main classes. They are shown in the class diagram.

5.1.1 DeviceConnector Class

Created to establish a connection with the REST — service JOC Cockpit on a specific device and exchange data with them. As an interface class provides the public method — the request. As an input method, the string — the text that should be placed in the body of the HTTP request. The value returned from the method is a string containing the body of the response from the device to the HTTP request.

The algorithm of the method is as follows:

- Send an authorization request.
- Get the access token needed to complete the rest of the requests (Auth method).
- Send a request by placing the received token in the headers and the text received as a method argument (SendRequest method) in the body.
- Receive a response from the device and return the contents of the response body.

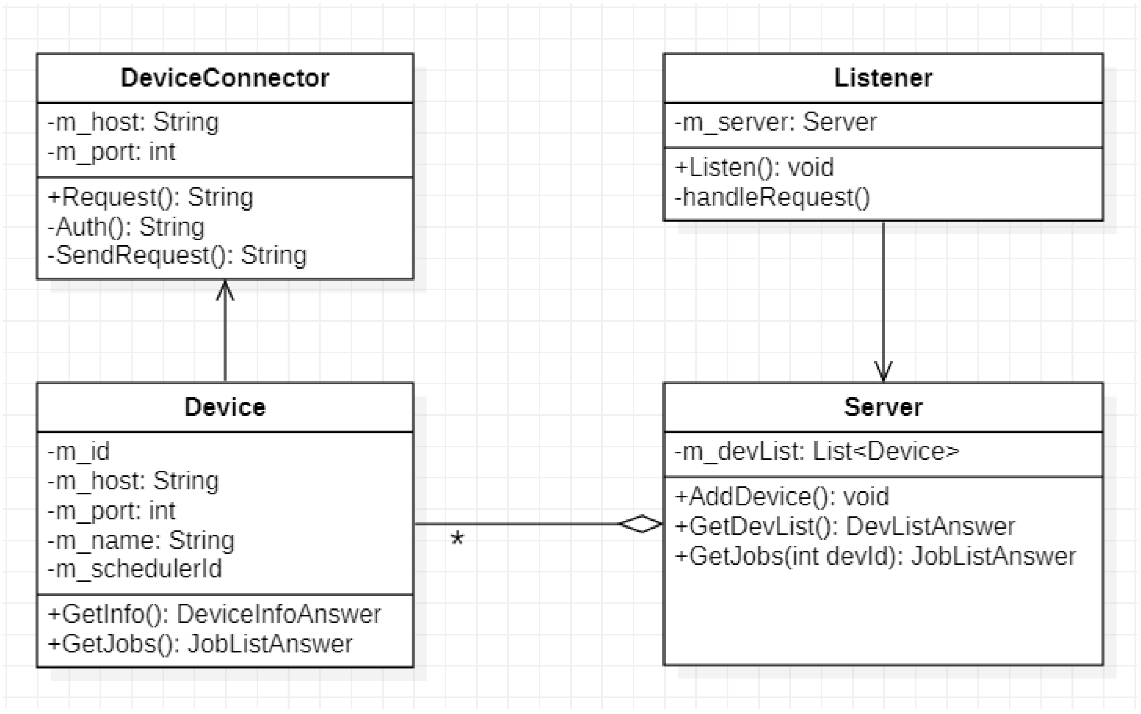


Figure 5.1: Server core classes

5.1.2 Device Class

It is an abstraction of the device that the server is monitoring. It stores the data necessary to establish a connection with this device. As an interface, it provides two public methods designed to receive device data:

- GetInfo is a method that returns information about the device: the identifier assigned by the server when adding the device, the address, and port of the device, the identifier of the task scheduler on the device, the device name assigned to display, as well as the current state of the scheduler on this device. Scheduler status is requested over the network using the DeviceConnector class.

- GetJobs is a method that allows getting a list of tasks created on the device, as well as information about each of the tasks. Information is also requested from the device over the network using the DeviceConnector class.

The described methods work according to the following algorithm:

- Formation from the data received from the client of the corresponding XML-request.
- Creating a new object of the Device Connector class, passing parameters for connection to it, calling the Request method, passing the generated request to it.
- Analysis obtained from the method Request response, parse its XML — structure extracting necessary data and returns.

5.1.3 Server Class

Stores a list of objects of the Device class — that is, information about all monitored devices. It provides three public methods as an interface :

- `AddDevice` – get the information required for the connection, creates a new object of class Device, and stores it to the list. The method does not cause any network exchange but stores the information necessary for network interaction with the device. Each device added to the list is assigned an integer identifier;
- `GetDevList` – returns a list of monitored devices, previously having requested each of them for status information by calling the GetInfo method;
- `GetJobs` – receives the device identifier at the input, finds the desired one in the list, and requests information about tasks from it by calling the GetJobs method.

The algorithm of the last two methods is straightforward. It consists of calling the corresponding methods of objects of the Device class, passing them the received data, and returning the responses received from them.

5.1.4 Listener Class

This class is designed for the HTTP — requests to parse, generate, and send responses to clients.

The Listen method is launched in the main thread of the program, and works, waiting for new requests from clients. Waiting for new requests is performed in a cycle, which is the primary cycle of the server program (English main program loop). The method works according to the following algorithm:

- Get a new HTTP request.

- Create a new thread in which the handleRequest method will be called that processes the request.
- Return to the beginning of the cycle.
- The handleRequest method works according to the following algorithm.
- XML parsing — the structure of the command received in the request.
- If an error is detected in the request, send an error message as a response to the client.
- Extract the necessary command parameters from the request.
- Call the method of the Server object of the class corresponding to the command, passing the parameters extracted from the request to it.
- Get the answer from the class object Server, generate the XML — representation of the response, and send it to the client.

The sequence diagram shows a diagram of the interaction of all server components when processing a single request from a client.

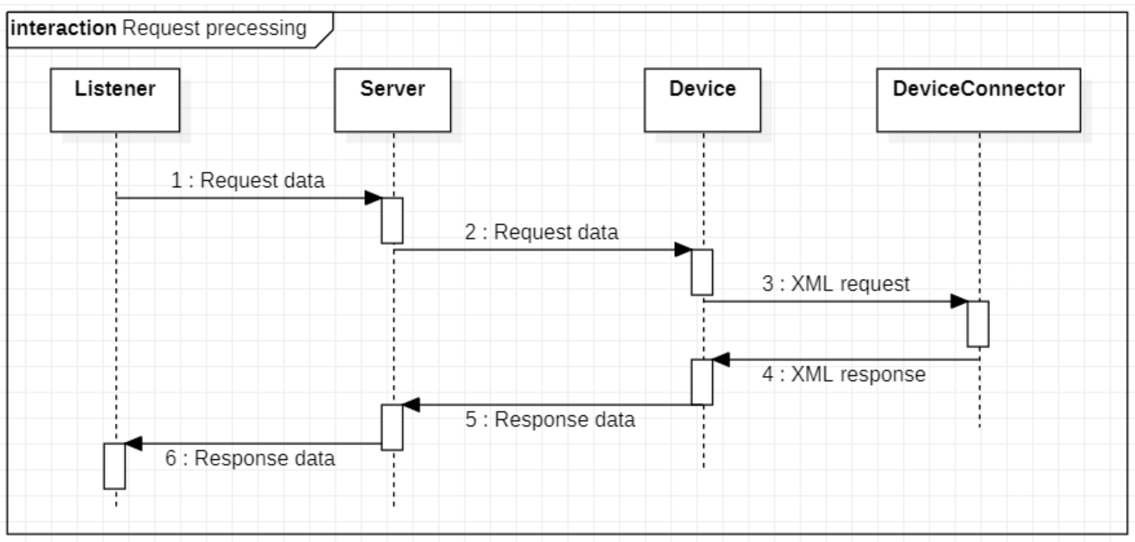


Figure 5.2: Sequence Diagram: Request Processing

The diagram shows that first, the requested data is first transmitted in a chain from component to component, then the received response and its data are transmitted in a chain back from the device to the client.

Thus, the server architecture is a set of layers, where the Listener class is located at the highest level, and DeviceConnector is located at the lower level. Each level requests a service from a layer of a lower level, passing data to it and receiving a response.

5.2 Formation of the XML — Answers

The diagram shows that some methods to obtain any information from the devices, return the Class objects DeviceInfoAnswer, DevListAnswer, JobAnswer, JobListAnswer. These classes implement the IAnswer interface with one method, GetXml.

The objects of these classes contain fields that store the data necessary to form the response. The GetXml method from this data forms the correct XML — the response corresponding to the request.

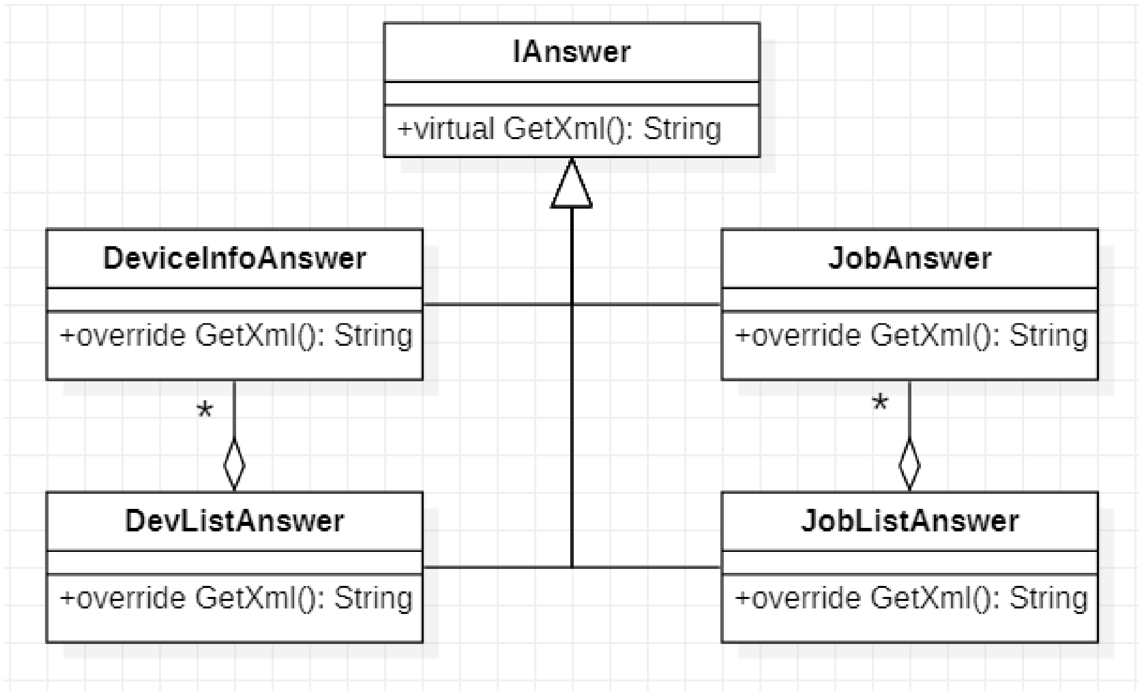

IAnswer
+virtual GetXml(): String
DeviceInfoAnswer
+override GetXml(): String
JobAnswer
+override GetXml(): String
*
*
DevListAnswer
+override GetXml(): String
JobListAnswer
+override GetXml(): String

Figure 5.3: Class diagram for forming a response to a request

The Listener class, receiving one of the objects implementing the IAnswer interface in response to the request, calls the GetXml method to generate a valid XML response, which is then sent to the client.

5.3 Additional Server Functionality

If we need to expand the list of data received from devices through the server or add capabilities for managing schedulers on devices, the current architecture allows us to make the appropriate changes to the code. To make such changes, we will need to do the following:

- Create new classes that implement the IAnswer interface to form new types of responses to the client.
- It is necessary to add methods to the Device class that will generate new types of requests for the device before passing them to DeviceConnector.

- In the Server class, we need to add new methods corresponding to the new types of client requests that will call the corresponding methods of the Device class and pass data to them.
- It is necessary to add new methods to the Listener class, which will parse the XML structure of the new types of client requests and call the corresponding methods of the Server class, passing the parameters extracted from the request into them.

Conclusions

In the course of the work, a review of existing single-board computers was conducted, an analysis of their characteristics was performed to select the appropriate one.

As a result of the study, a single-board Colibri computer was selected because it has the necessary properties: OS and open source drivers, regular updates, and is small in size.

The review of existing task schedulers was conducted to implement a task management system on devices. As a result of the analysis, the JobScheduler was selected, since it is multi-platform and provides the network API, which is necessary for implementing remote monitoring and scheduler management.

In the course of this work, a prototype server was developed that allows monitoring the status of tasks on devices with the JobScheduler scheduler installed. The server provides the ability to connect to many devices, monitor the status of the task scheduler for them, receive information about tasks created on the devices: name and title, whether the task is enabled, its status, next run time, source code of the script that the task starts, script language. The server provides a web-based REST API through which it interacts with the server. Client applications send XML commands to the server. The server in the process of processing requests the necessary data from the devices and returns the received information to the client as a response.

In Chapter 4, a real situation was simulated in which the task scheduler received data from devices, processed each request. After collecting all the necessary data, the server provided the information the user needed, successfully handling tasks with an incoming data flow rate of up to 100 Mb/s with allows to run more than 100 jobs, working on Linux and Windows systems. Thanks to the use of the server and the JobScheduler, it was possible to create a solution used to monitor and manage tasks on equipment that will be located remotely and, at the same time, possible work with various operating systems. The server architecture allows to expand its functionality and add new scenarios of user interaction with end devices.

Bibliography

[1] PFISTER, Cuno. *Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud*. Maker Media, 2011. Available also from: https://www.amazon.com/Getting-Started-Internet-Things-Microcontrollers-ebook/dp/B00COVJUGI.

[2] SHARMA, Meenakshi. *Stages of "Internet of Things" Architecture* [online]. 2015 [visited on 02/20/2020]. Available from: https://www.marlabs.com/blog-stages-of-iot-architecture.

[3] SHARMA, Meenakshi. *Stages of "Internet of Things" Architecture* [online]. 2019 [visited on 02/20/2020]. Available from: https://www.marlabs.com/blog-stages-of-iot-architecture.

[4] AXMOR. *How IoT works: analysis of technical solutions* [online]. 2018 [visited on 02/20/2020]. Available from: https://axmor.ru/articles/tehnologiya-iot-analiz-tehnicheskih-reshenij.

[5] HAT, Red. *What is a Raspberry Pi?* [online]. 2019 [visited on 02/20/2020]. Available from: https://opensource.com/resources/raspberry-pi.

[6] MEDIAWIKI. *ZoneMinder Wiki* [online]. 2018 [visited on 02/20/2020]. Available from: https://wiki.zoneminder.com.

[7] PI, Orange. *What's Orange Pi?* [online]. 2018 [visited on 02/20/2020]. Available from: http://www.orangepi.org.

[8] TORADEX. *Quickstart Guide Torizon* [online]. 2019 [visited on 02/20/2020]. Available from: https://developer.toradex.com/products.

[9] RADXA. *Introduce the new ROCK Pi* [online]. 2018 [visited on 02/20/2020]. Available from: https://wiki.radxa.com.

[10] TORADEX. *Colibri iMX6* [online]. 2017 [visited on 02/20/2020]. Available from: https://developer.toradex.com/products/colibri-imx6.

[11] ALAPATI, Sam R. *Expert Oracle Database*. Apress, 2018. Available also from: https://www.amazon.com/Expert-Oracle-Database-Administration-Experts/dp/143021015X.

[12] NETAPPLICATIONS. *Browser Market Share* [online]. 2017 [visited on 02/20/2020]. Available from: https://netmarketshare.com.

[13] MICROSOFT. *Task Scheduler for developers* [online]. 2020 [visited on 02/20/2020]. Available from: https://docs.microsoft.com/cs-cz/windows/win32/taskschd/task-scheduler-start-page.

[14] ORGANISATIONS SERVICE, Software und. *Architecture of JobScheduler* [online]. 2020 [visited on 03/05/2020]. Available from: https://kb.sos-berlin.com/display/PKB/Architecture.

[15] HOGDAL, ScottJ. *Analysis and Diagnostics of Computer Networks*. Laurie, 2001.

[16] HABRAKEN, Joe. *Do yourself a wireless network*. Sams Publishing, 2016.

[17] PEREIRA ZAPATA, Omar U. *EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation* [online]. 2019 [visited on 01/25/2020]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.4620.

[18] BARD, John. *Programmable Radio Communication Network Architecture* [online]. 2019 [visited on 01/25/2020]. Available from: https://www.researchgate.net/publication/261180140_Software_Defined_Radio_The_Software_Communications_Architecture.

[19] SCHWARZ, Michael. *Linux Job Scheduling* [online]. 2000 [visited on 01/25/2020]. Available from: https://www.linuxjournal.com/article/4087.

[20] BELL, Charles. *Windows 10 for the Internet of Things* [online]. 2017 [visited on 01/25/2020]. Available from: https://www.apress.com/gp/book/9781484221075.

[21] SIMMONDS, Chris. *Mastering Embedded Linux Programming* [online]. 2015 [visited on 01/25/2020]. Available from: https://subscription.packtpub.com/book/networking_and_servers/9781784392536.

[22] GONZALEZ, Alex. *Embedded Linux Development Using Yocto Project Cookbook* [online]. 2018 [visited on 01/25/2020]. Available from: https://subscription.packtpub.com/book/virtualization_and_cloud/9781788399210.

[23] MADIEU, John. *Linux Device Drivers Development* [online]. 2017 [visited on 01/25/2020]. Available from: https://subscription.packtpub.com/book/networking_and_servers/9781785280009.

Index

Appendix

A Colibri iMX6

Toradex recently introduced the Linux-driven Colibri iMX6ULL, one of the first modules to adopt the new, optimized in terms of power and cost SoC “i.MX6 ULL” from NXP. The new Colibri module also supports dual-band WiFi-AC and BT 4.2. On March 28, 2017, Toradex announced the Colibri iMX6ULL computer module at Embedded World. As its name suggests, this new member of the Toradex Colibri COM 67.6 x 36.7 mm family combines the recently announced IXM ULL SoC from NXP, an ultra-low-power option of the ubiquitous NXP i.MX6. The new COM is also the first Toradex Colibri module to include integrated wireless connectivity. Comparison of NXP i.MX6 ULL block diagrams (Figure 1) and i.MX6 UL SoC (Figure 2).

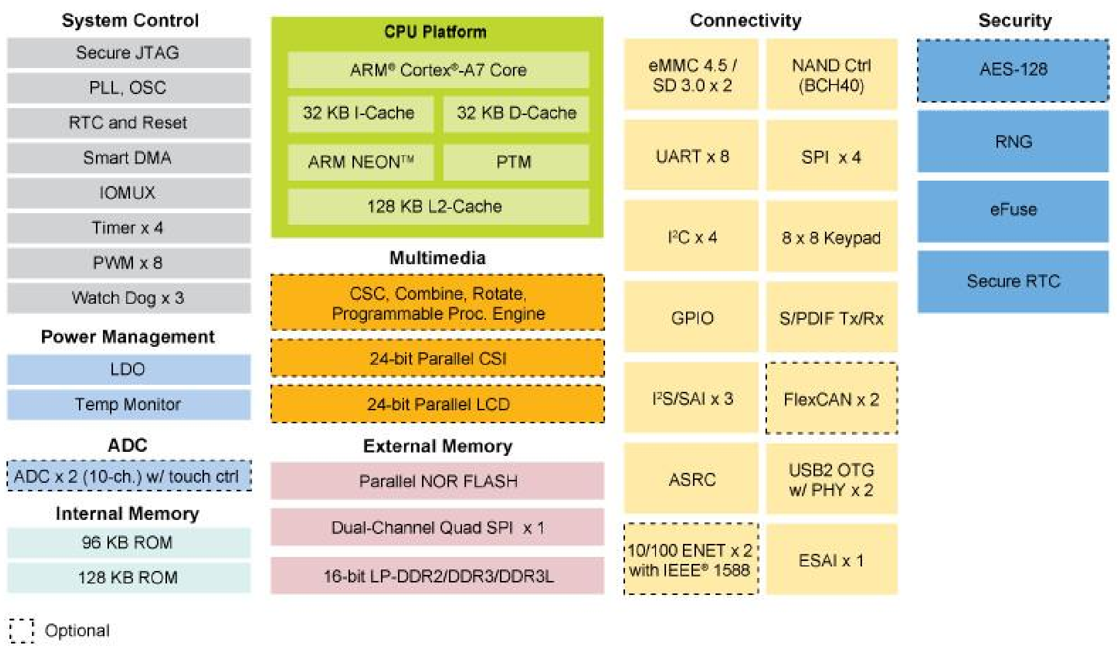


Figure 1: NXP i.MX6 ULL [9]

New features in i.MX6 ULL compared to i.MX6 UL includes multi-channel audio input/input ESAI and 24-bit (or preferably 16-bit) CSI interface. Besides, the

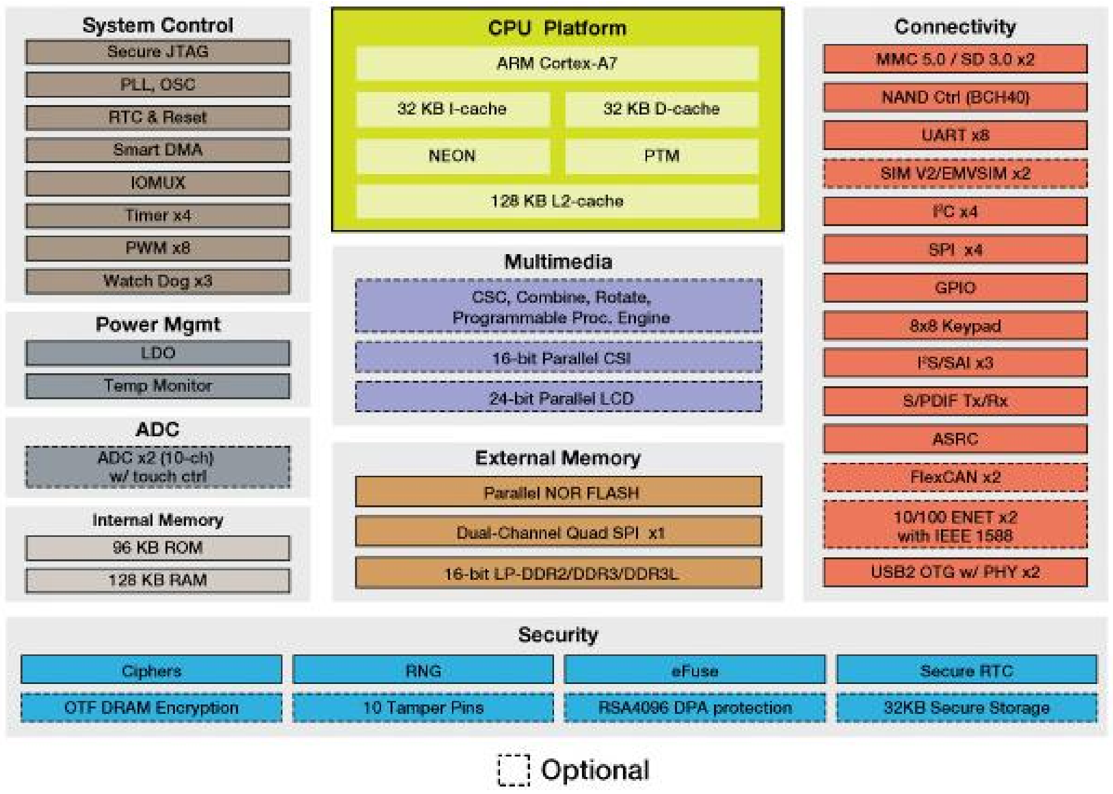


Figure 2: i.MX6 UL SoC [9]

EMV SIM UL module has been removed to expand cellular radio communications. Additional features in ULL extend beyond the ADC and UL multimedia interfaces and include 10/100 Ethernet and FlexCAN. ULL also adds an EPD / PXP interface for the e-book market (Figure 3).

Both UL and ULL provide new security, unauthorized access, and power management features not found in earlier i.MX6 models, while ULL offers more highly optimized, low-power modes than UL. Although ULL continues to offer basic security features such as secure boot, hardware cryptographic encryption mechanisms, and random number generators, it removes those UL security features that are targeted at POS applications, including CAAM / BEE / DryICE. Like UL, ULL offers a stripped-down, but still 3D-capable, WXGA display compared to the more powerful Vivante GPUs from earlier i.MX6 SoCs. The Colibri iMX6ULL is the first of 67.6 x 36.7 mm COM Toradex to use NXP's new Socket i.MX6 ULL SoX, as well as the first Colibri module, to include integrated wireless connectivity. It follows the Colibri i.MX6 based on i.MX6, among other Toradex modules (Figure 4).

Toradex offers the Colibri iMX6ULL in two standard SKUs: an 800 MHz clock board, 512 MB DDR3L SDRAM and 512 MB SLC flash memory and a cheaper model with 528 MHz clock speed, 256 MB RAM and 128 MB flash memory. Also, the first of them includes an integrated wireless subsystem that supports Bluetooth 4.2 with BLE, as well as dual-band WiFi with a transfer speed of up to 300 Mbps, as well as simultaneous station and access point modes (Figure 5).

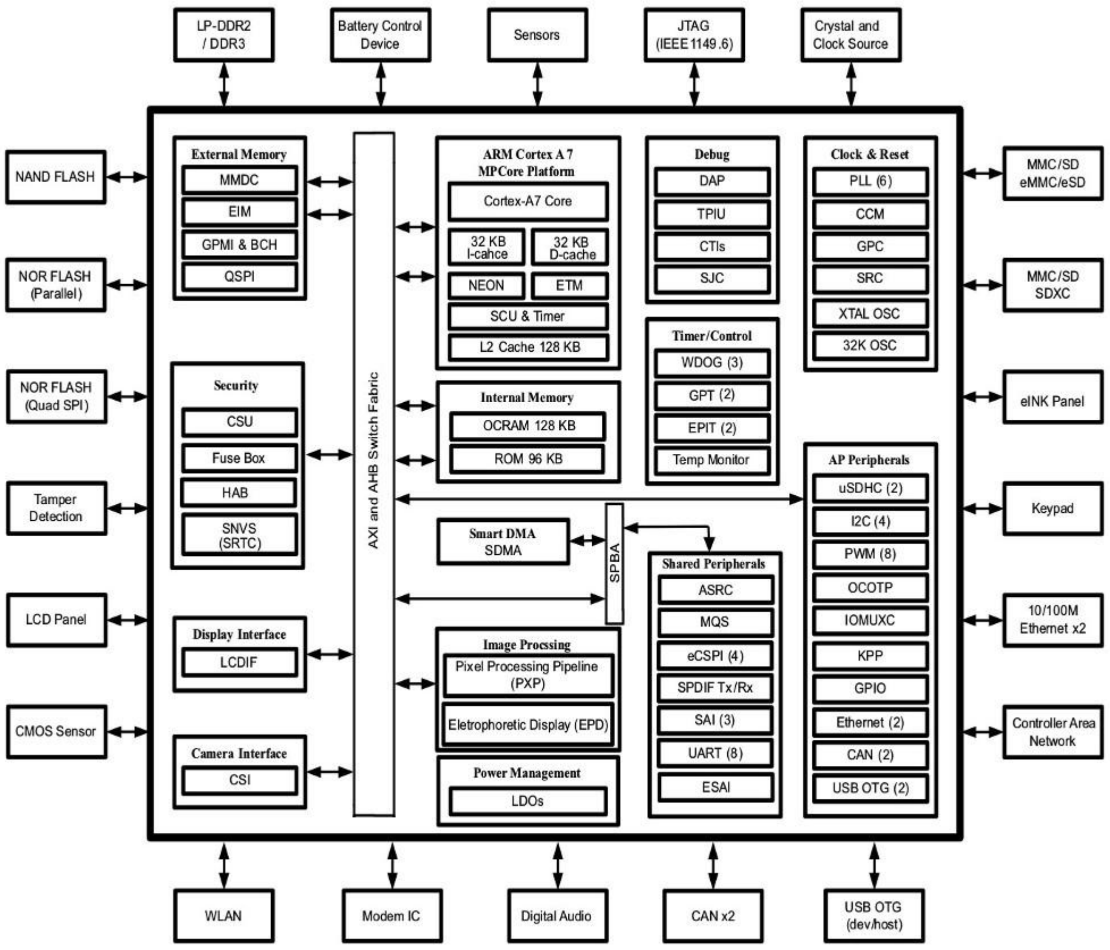


Figure 3: Colibri iMX6ULL interface [10]

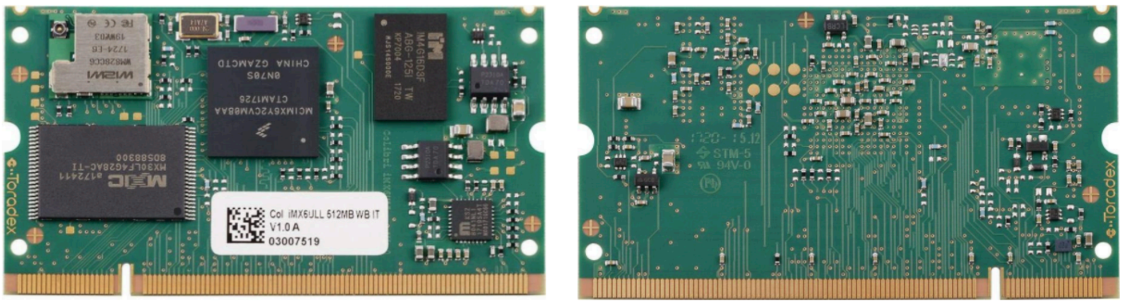


Figure 4: Colibri iMX6ULL front and back view [9]

Other features common to both Colibri iMX6ULL models include a USB host port and OTG ports, an Ethernet 10/100 controller with support for up to two interfaces, SDIO / SD / MMC, a 24-bit RGB LCD interface with 4-wire resistive touch, serial camera interface and stereo digital sound. The Colibri iMX6ULL is additionally equipped with an extensive set of SDIO, I2C, SPI, UART, PWM, and CAN interfaces, as shown in the block diagram above. The 67.6 x 36.7 x 6.2 mm mod-

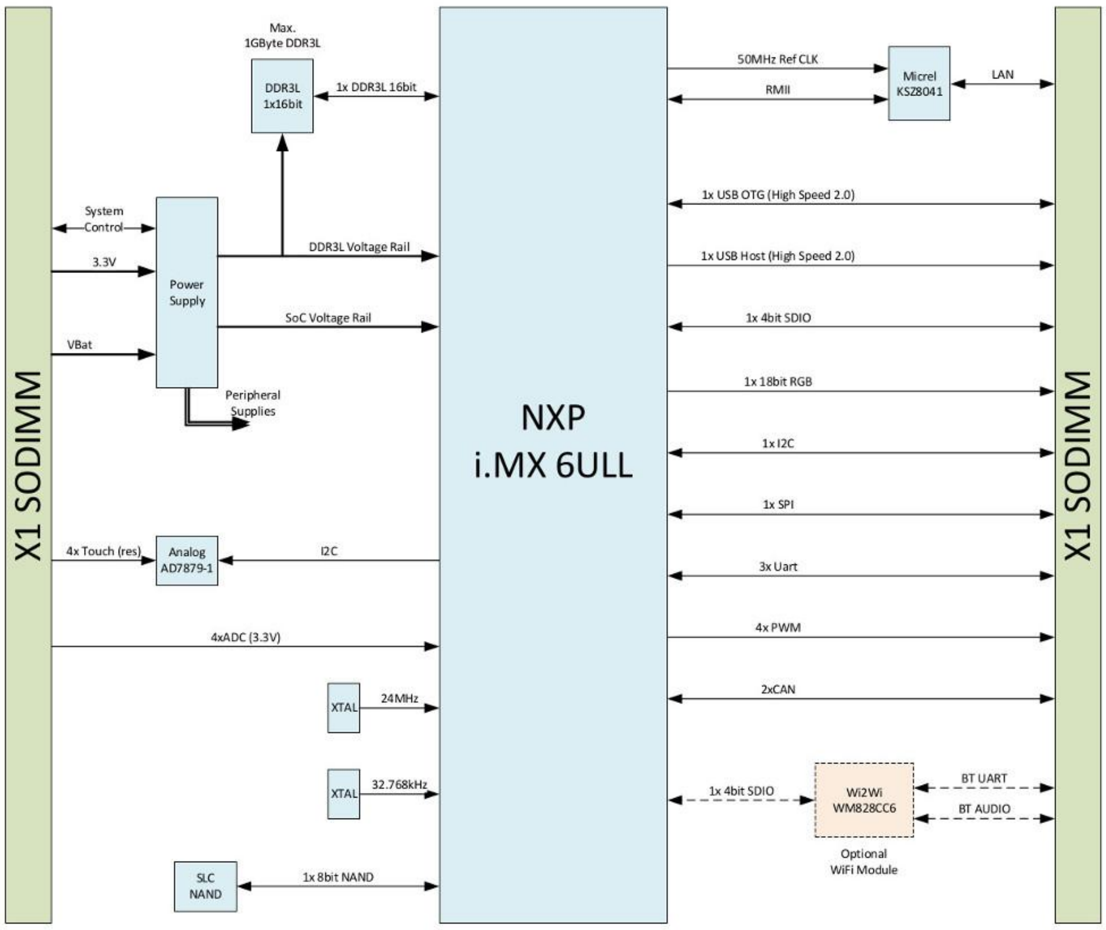


Figure 5: Colibri iMX6ULL block diagram [9]

ule is available in versions from 0 to 70 °C and -40 to 85 °C and provides minimum availability until 2028. The module comes with a Linux distribution based on NXP's Yocto Project. The Colibri iMX6ULL comes at a low price of 34.75$ (with 256 MB / 128 MB of RAM / flash) and 58.60$ (512 MB / 512 MB of RAM / flash + wireless) [9].

B Installation of JobSheduler environment

B.1 Database Server Setup

JobScheduler stores all data about tasks on the database server, which can be located both on the remote node and locally. For work, it is required that for the purposes of JobScheduler, a separate database be created on the server, as well as a user with full rights to it. Database tuning is described here using the example of MySQL.

The first thing to do is to create a new database:

```
mysql> CREATE DATABASE scheduler;
```

```
Query OK, 1 row affected (0.01 sec)
```

Then, create a user under whose name JobScheduler will have to log in.

```
mysql> CREATE USER scheduler IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.01 sec)
```

The user needs full rights to work with the created database.

```
mysql> GRANT ALL PRIVILEGES ON scheduler. * TO scheduler;
Query OK, 0 rows affected (0.01 sec)
```

In these listings, we should pay attention to the selected database and user names — scheduler, as well as the user password — 1234. These data will be needed in subsequent steps.

It should be noted that the names for the database and the user scheduler — are selected merely as an example. Instead of these names, any others can be selected. However, these data must be remembered.

B.2 Installation and Configuration of JobScheduler

The scheduler is distributed freely and is available on the official JobScheduler project website as an archive.

The archive contains, among other things, a script designed to start the installation process. On Windows, the file is called setup.cmd; on Linux, the file is called setup.sh. Running this script will launch the installation wizard.

The wizard provides a reasonably simple dialog interface. At each stage of installation, the user is asked standard questions. In this text, attention is paid to the installation steps that relate to the scheduler settings.

After specifying the directory in which the application should be installed (installation path), you will need to specify the directory in which the scheduler configuration files will be placed, as well as all the logs (path for configuration files and logs). It should be noted that these should be two different directories.

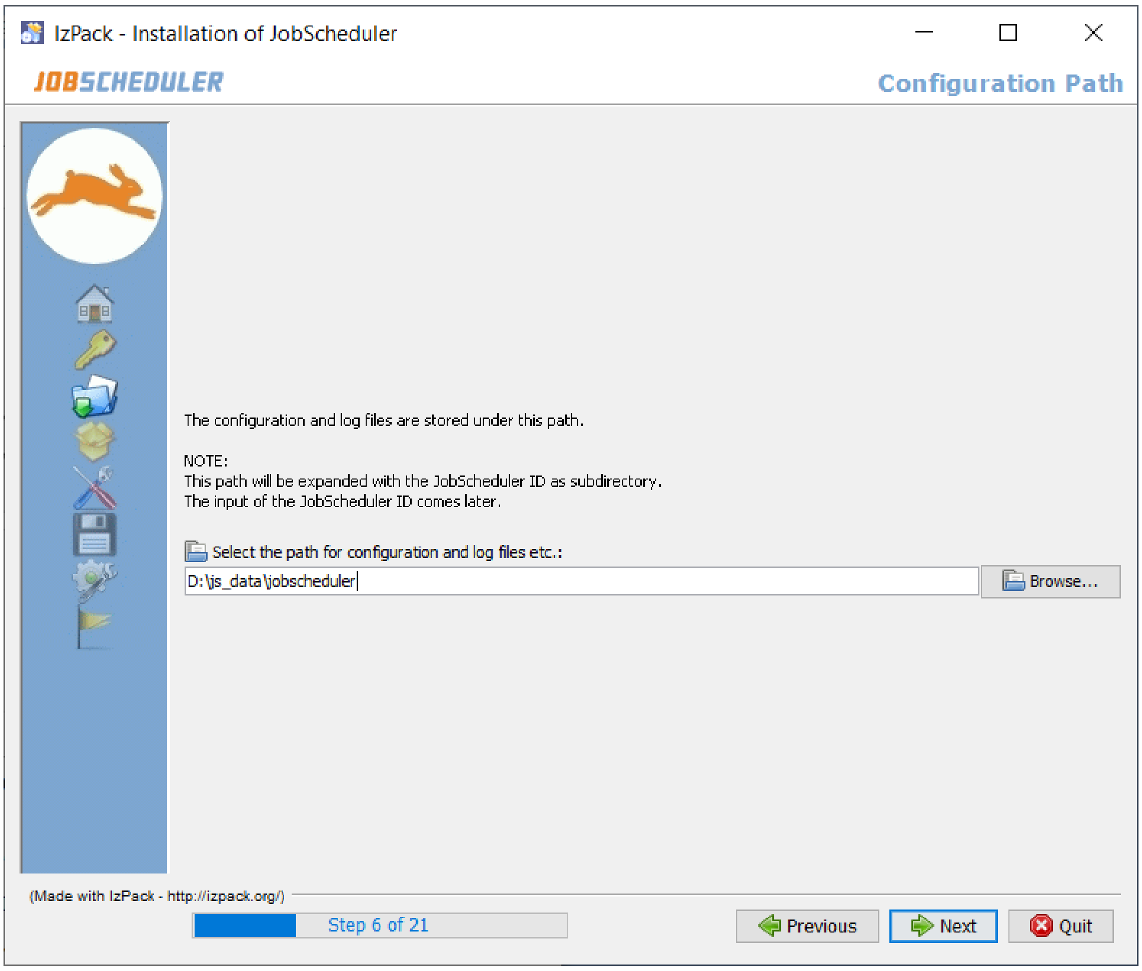
Since JobScheduler is run on behalf of a regular OS user who does not have administrator rights, the directory with the configuration and logs should be located in the area of the file system to which this user has access. In the case of Linux, this should be the user's home directory.

The next step is to specify the data such as the HTTP Scheduler port, as well as the site address, from which should be allowed access to JobScheduler.

The figure highlights information that we can specify at discretion but must be remembered. This is the port that the scheduler will listen to via the HTTP protocol, as well as the identifier of the JobScheduler process on this node.

The identifier must be remembered, as it will be required when accessing the scheduler via the REST API.

The address of the allowed node must be entered for security reasons. It is believed that when the scheduler can receive requests from one specific node, this has a positive effect on security.


IzPack - Installation of JobScheduler

JOBSCHEDULER

Configuration Path

The configuration and log files are stored under this path.

NOTE:
This path will be expanded with the JobScheduler ID as subdirectory.
The input of the JobScheduler ID comes later.

Select the path for configuration and log files etc.:

D:\js_data\jobscheduler

Figure 6: Request directory for installation

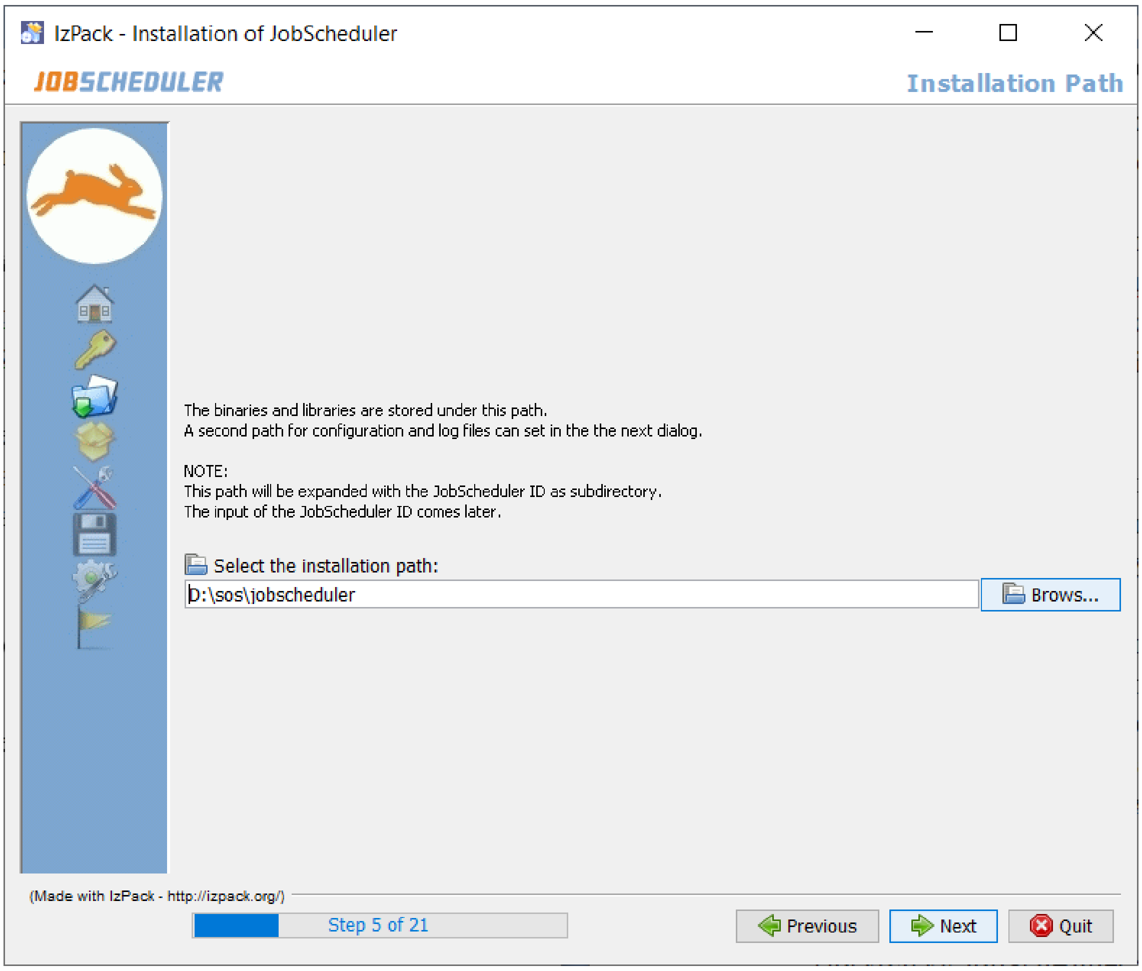
However, if the scheduler is required to be accessible from any node, the IP-address 0.0.0.0 should be specified in this field.

The following installation steps ask for data to connect to the database. The figure shows an example of setting access to the database server. Since, in this case, the scheduler configuration is considered using the example of MySQL, this system is chosen as the DBMS (database management system).

The next step involves entering data to authorize JobScheduler on the database server. Here we need to enter precisely the data that was selected at stage 2. The user name is scheduler, the database name is scheduler, and the user password is 1234.

The "port" field indicates the standard port on which the MySQL server is running. If the server we are using is configured to work with another port, then the data in the field should be changed accordingly.

In addition, it should be noted that the "Host" field is automatically filled in by the wizard. The hostname is the name of the computer on which the installation is performed. If it is assumed that the database server is running locally, then it is advisable to change the value in this field to "localhost", since the computer may


IzPack - Installation of JobScheduler

The binaries and libraries are stored under this path.
A second path for configuration and log files can set in the the next dialog.

NOTE:
This path will be expanded with the JobScheduler ID as subdirectory.
The input of the JobScheduler ID comes later.

Select the installation path:
D:\sos\jobscheduler

(Made with IzPack - http://izpack.org/)

Step 5 of 21

Figure 7: Request a directory of configuration files and logs

not be configured to access by network name.

At the next step, the wizard reports that it is desirable to use the same database for JobScheduler and JOC Cockpit.

In the event that the adjustment is performed in this way, the attached checkbox should be marked.

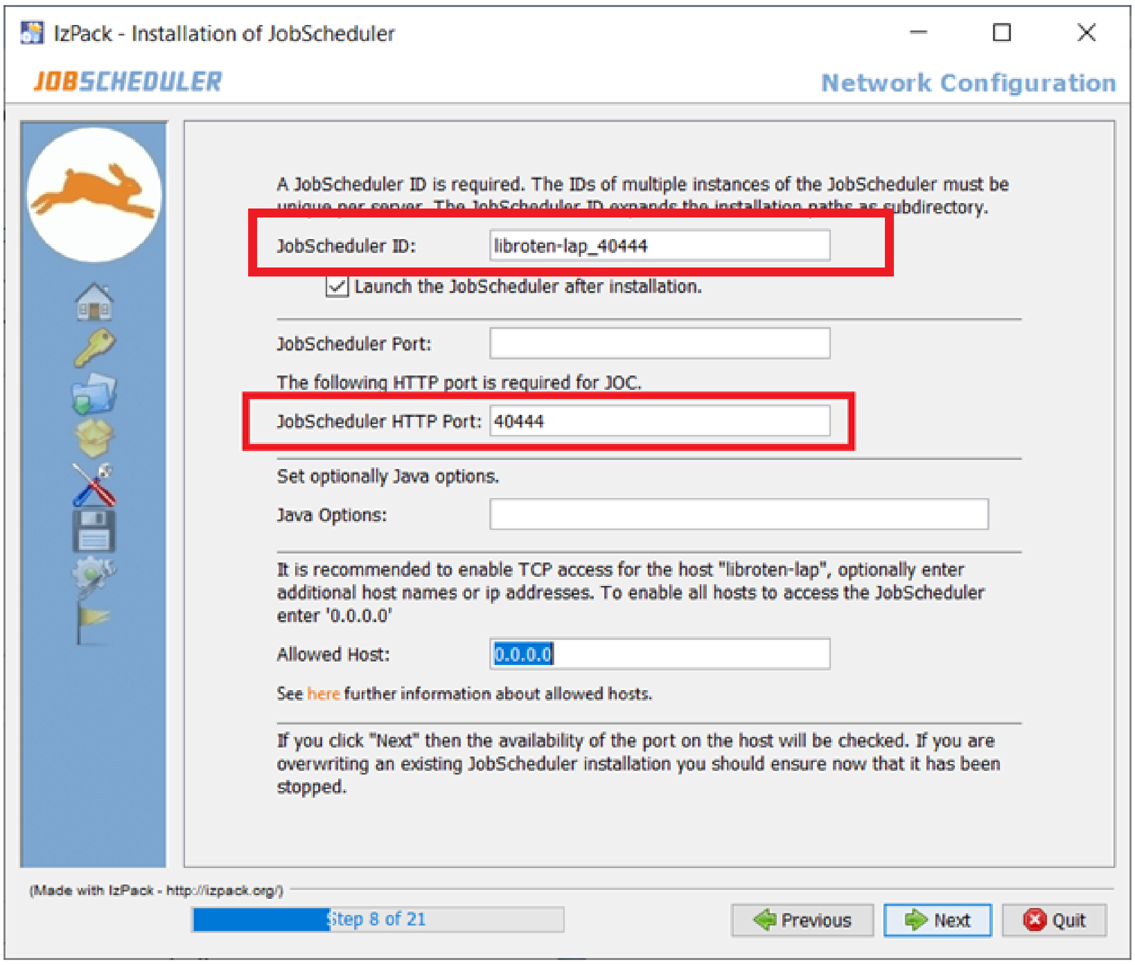
After clicking on the “Next” button, the automatic installation process starts.

B.3 Installation and Configuration of JOC Cockpit

JOC Cockpit software is free and available on the developer’s website.

The software is offered for download in the form of an archive in which, in the same way as in the case with JobScheduler, in addition to the installation files, there are a setup script.cmd or setup.sh (depending on the operating system used), which launches the installation wizard.

The installation process is similar. Let us dwell on the steps related to setting up the JOC Cockpit.


IzPack - Installation of JobScheduler

JOBSCHEDULER

Network Configuration

A JobScheduler ID is required. The IDs of multiple instances of the JobScheduler must be unique per server. The JobScheduler ID expands the installation paths as subdirectory.

JobScheduler ID: libroten-lap_40444

☑ Launch the JobScheduler after installation.

JobScheduler Port:

The following HTTP port is required for JOC.

JobScheduler HTTP Port: 40444

Set optionally Java options.

Java Options:

It is recommended to enable TCP access for the host "libroten-lap", optionally enter additional host names or ip addresses. To enable all hosts to access the JobScheduler enter '0.0.0.0'

Allowed Host: 0.0.0.0

See here further information about allowed hosts.

If you click "Next" then the availability of the port on the host will be checked. If you are overwriting an existing JobScheduler installation you should ensure now that it has been stopped.

(Made with IzPack - http://izpack.org/)

Figure 8: Specifying data to connect JobScheduler

At the beginning of the installation, in the same way, as in the case of JobScheduler, it is proposed to specify a directory for placing program files and a directory for placing configuration files and logs.

In the next step, the wizard prompts us for the Jetty software installation options. Jetty is a web server designed specifically for organizing a REST API for interacting with JobScheduler remotely over a network.

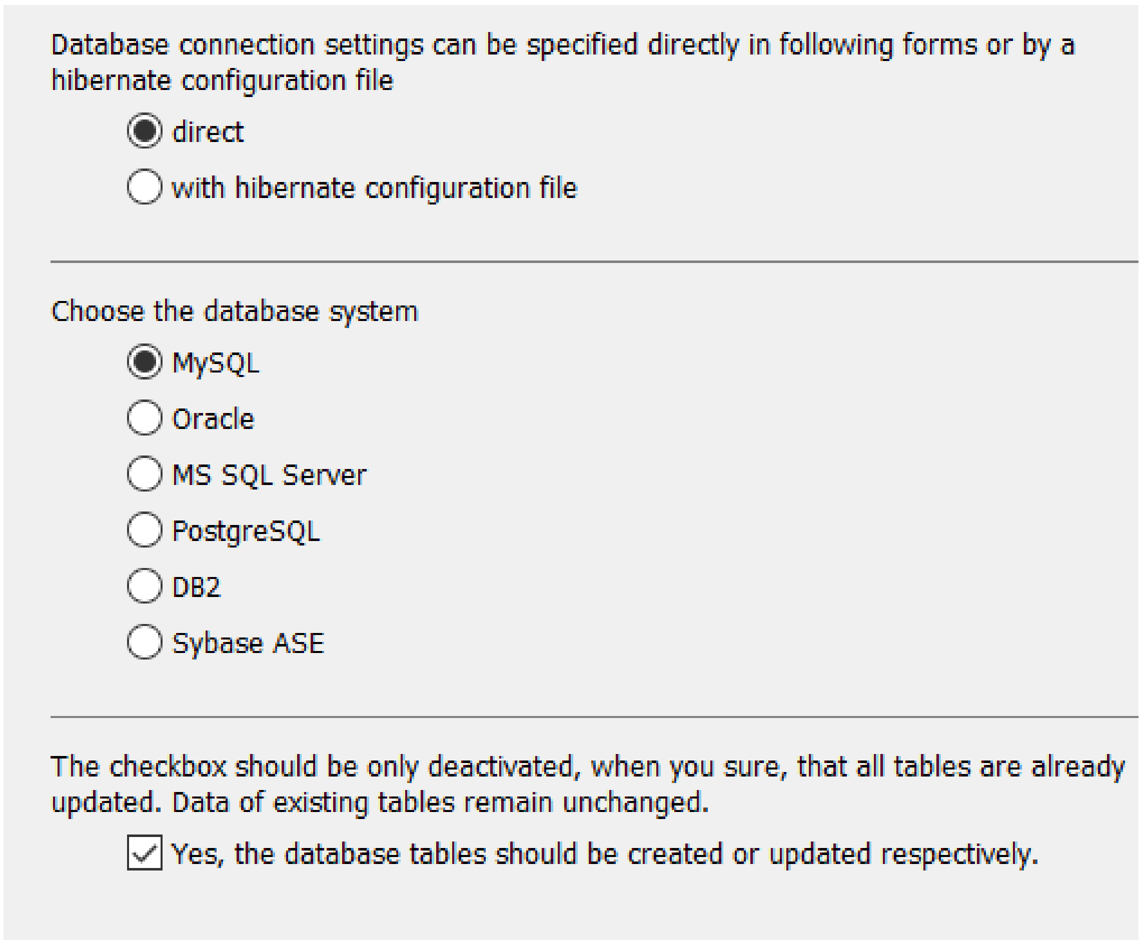
JOC Cockpit allows its use with a third-party web server. However, this will require additional configuration of both preferred web server and JOC.

At this step, pay attention to the port on which Jetty will run. It can be changed to any other necessary, but it should be remembered since it is through this port that the interaction with JobScheduler through the REST API will take place.

The next step is to configure the connection to the database. Data is entered similar to those used in setting up JobScheduler.

Need to specify the same database name, user name, and user password — the data that was selected in step 2. In this example, the following data is used. The database name is scheduler, the user name is scheduler, the password is 1234.

As in the case of JobScheduler, the localhost value should be specified in the


Database connection settings can be specified directly in following forms or by a hibernate configuration file

[✓ direct]

with hibernate configuration file

Choose the database system

[✓ MySQL]

Oracle

MS SQL Server

PostgreSQL

DB2

Sybase ASE

The checkbox should be only deactivated, when you sure, that all tables are already updated. Data of existing tables remain unchanged.

[✓ Yes, the database tables should be created or updated respectively.]

Figure 9: DBMS selection

“host” field if the database server is running locally — on the same node as the JOC.

B.4 Launch the Required Services

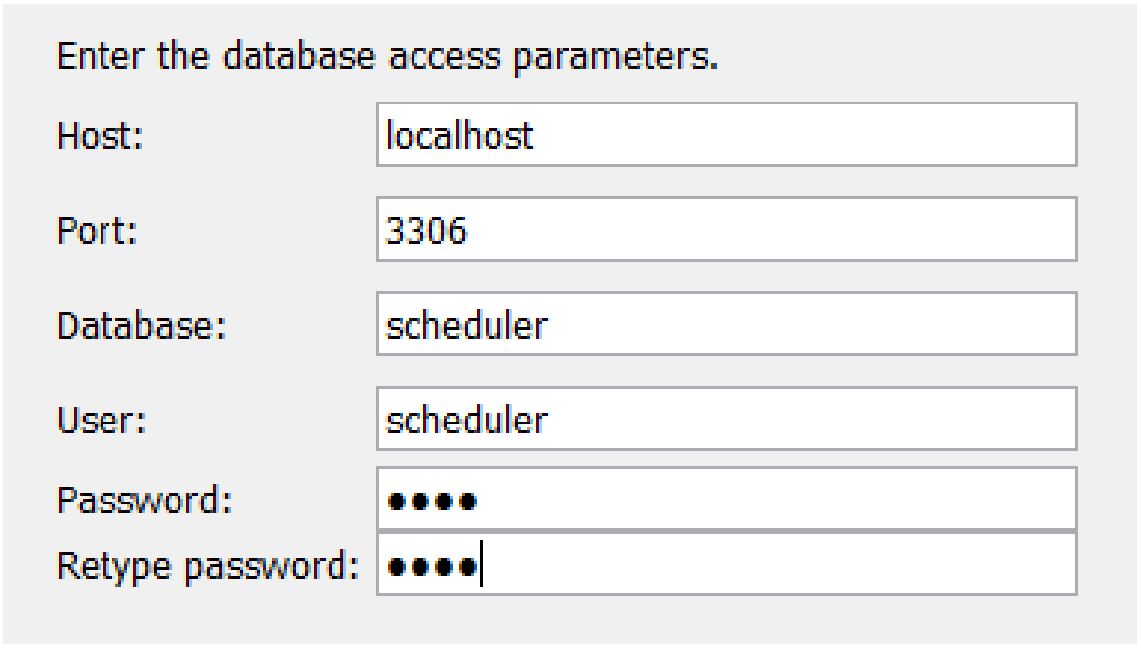
For the scheduler infrastructure to work, the following services must be running:

- JobScheduler.
- JOC Cockpit.
- Jetty (or other selected webserver).

JOC and Jetty are installed on the system, and then registered as services, and start automatically.

JobScheduler, in turn, is a process that is launched through an executable file located in the directory in which the scheduler was installed.

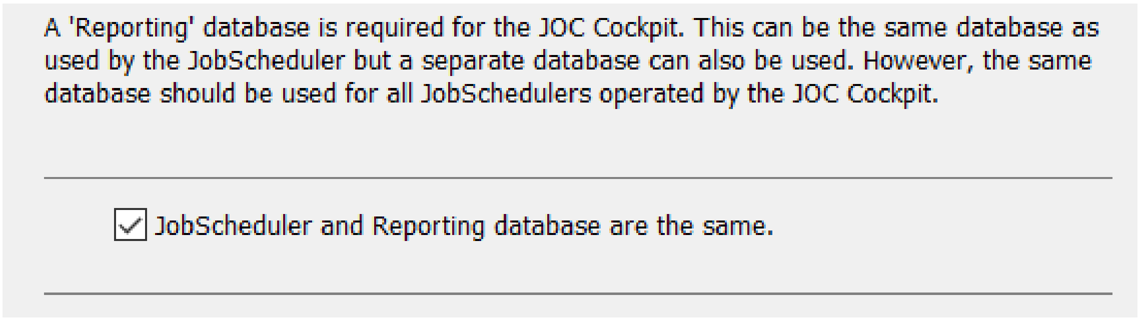
This executable file is located on the system in the following path:



Enter the database access parameters.

Host:	localhost
Port:	3306
Database:	scheduler
User:	scheduler
Password:	●●●●
Retype password:	●●●●

Figure 10: Data for connecting to the database server



A 'Reporting' database is required for the JOC Cockpit. This can be the same database as used by the JobScheduler but a separate database can also be used. However, the same database should be used for all JobSchedulers operated by the JOC Cockpit.

[✓] JobScheduler and Reporting database are the same.

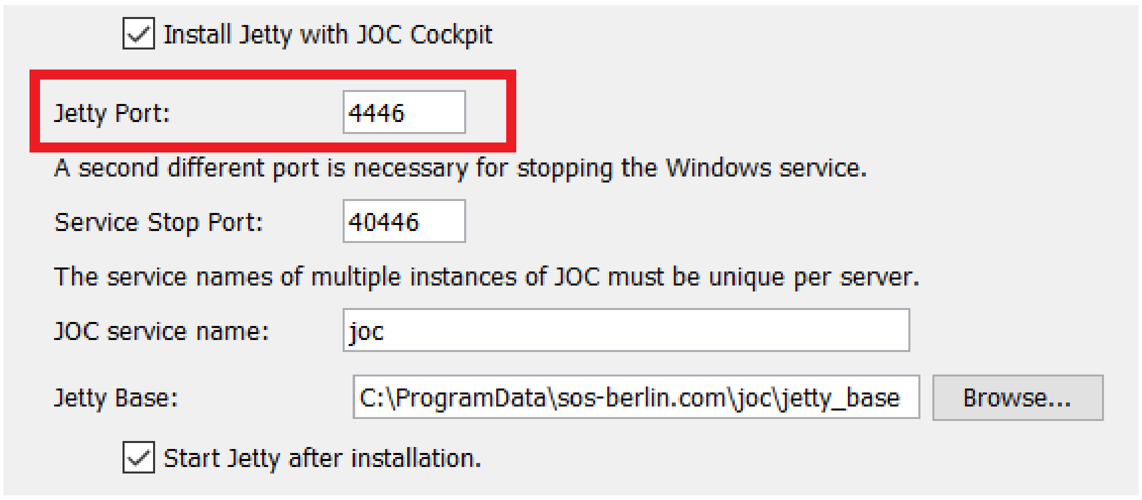
Figure 11: Configuring JobScheduler and JOC Cockpit to work with one database

```
<JobScheduler directory>/bin/jobscheduler.cmd
(extension can be *.sh if using Linux OS).
```

This script requires passing command line arguments, so we should run it through the terminal (shell for Linux, CMD, or PowerShell for Windows), after moving to the above directory.

Starting the JobScheduler process is done using the command:

```
joscheduler.cmd start
```


☑ Install Jetty with JOC Cockpit

Jetty Port: 4446

A second different port is necessary for stopping the Windows service.

Service Stop Port: 40446

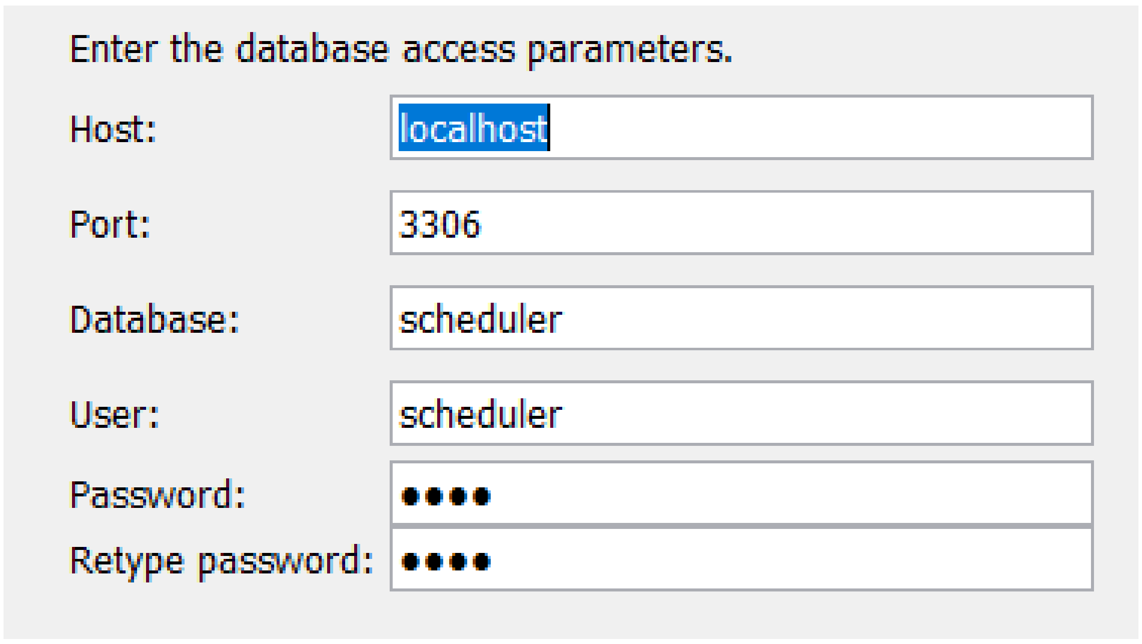
The service names of multiple instances of JOC must be unique per server.

JOC service name: joc

Jetty Base: C:\ProgramData\sos-berlin.com\joc\jetty_base

☑ Start Jetty after installation.

Figure 12: Jetty Setup


Enter the database access parameters.

Host:	localhost
Port:	3306
Database:	scheduler
User:	scheduler
Password:	••••
Retype password:	••••

Figure 13: Configure JOC database connection

C JobScheduler commands

Table 1: JobScheduler commands

Title of the commands	Describes
<terminate/>	Terminates a JobScheduler Master
<start_job/>	Starts a job
<show_state/>	Retrieves status information about a JobScheduler Master
<show_order/>	Retrieve order status information
<show_job_chain/>	Retrieves status information about a job chain
<show_job/>	Retrieves status information about a job
<show_history/>	Retrieve the JobScheduler history
<remove_order/>	Removes a temporary ad hoc order from the JobScheduler Master
<params.get/><param.get name="..."/>	Shows parameters from ./config/scheduler.xml
<modify_spooler cmd="stop"/>	Stops any running tasks in a JobScheduler Master
<modify_spooler cmd="pause"/>	Pauses a JobScheduler Master
<modify_spooler cmd="continue"/>	Continues operation of the JobScheduler Master after pausing
<modify_spooler cmd= "abort_immediately_and_restart"/>	Aborts and restarts a JobScheduler Master
<modify_spooler cmd= "abort_immediately"/>	Aborts a JobScheduler Master
<modify_order/>	Updates order attributes
<modify_order suspended="yes"/>	Suspends an order
<modify_order state="..."/>	Modifies an order's state
<modify_order action="reset"/>	Resets an order
<modify_job cmd="unstop"/>	Unstops a previously stopped job
<modify_job cmd="suspend"/>	Suspends all running tasks for a job
<modfiy_order at="now"/>	Causes an immediate start of the order
<kill_task/>	Kills a task that is running for the job
<job_chain_node.modify action="stop"/>	Stops a job chain node
<job_chain.modify state="stopped"/>	Stops a job chain
<job_chain.modify state="running"/>	Continues a previously stopped job chain
<add_order/>	Add a temporary order to a job chain

D Windows Task Scheduler example

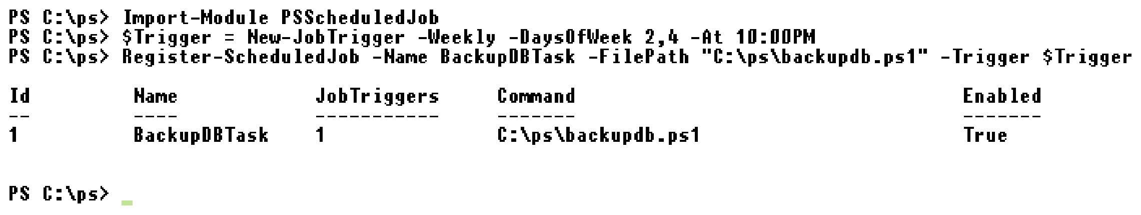
Suppose we need to create a task that will run a specific script every Monday and Thursday at 11:00. To create a new job, use the cmdlets of the PSScheduledJob module. For this cmdlet to work, you must install at least PowerShell version 3.0. Run the PowerShell console with administrator privileges and import the PSScheduledJob module.

Create a new trigger:

```
$Trigger = New-JobTrigger -Weekly -DaysOfWeek 1,4 -At 11:00PM
```

Now create a new task called BackupDBTask and bind to it the trigger created earlier (Figure 14):

```
Register-ScheduledJob -Name BackupDBTask -FilePath
"C:\ps\backupdb.ps1" -Trigger $Trigger
```



```
PS C:\ps> Import-Module PSScheduledJob
PS C:\ps> $Trigger = New-JobTrigger -Weekly -DaysOfWeek 2,4 -At 10:00PM
PS C:\ps> Register-ScheduledJob -Name BackupDBTask -FilePath "C:\ps\backupdb.ps1" -Trigger $Trigger

Id         Name            JobTriggers     Command                                  Enabled
--         ----            -----------     -------                                  -------
1          BackupDBTask    1               C:\ps\backupdb.ps1                       True

PS C:\ps> _
```

Figure 14: Creating a new Task Scheduler

Scheduler task created. You can find it in the Task Scheduler graphical console in the section: Task Scheduler -> Task Scheduler Library -> Microsoft-> Windows -> PowerShell -> SheduledJobs. When creating a new job, PowerShell generates a new XML file that contains the job definition. You can find this file in the profile of the current user:

```
%USERPROFILE%\AppData\Local\Microsoft\Windows\PowerShell\
ScheduledJobs\BackupDBTask
```

In the future, this XML file can be used to import job settings on other computers. Import (creating a task with settings from XML) is performed by the following command:

```
Register-ScheduledTask -Xml (Get-Content
'\\srv1\ScheduledJobDefinition.xml' | out-string)
-TaskName "BackupDBTask"
```

In Windows Server 2012 and later, when creating a scheduler job, importing the PSScheduledJob module is not necessary. If you need to run a task from a specific user, you need to get his password (it will be saved in the password manager, which of course is not very secure):

```
$credent = Get-Credential contoso\server-admin1
```

To start the task with elevated rights, enable the RunElevated flag:

```
$elevat = New-ScheduledJobOption -RunElevated
```

Now create a new task:

```
Register-ScheduledJob -Name BackupDBTask2 -FilePath
C:\ps\backupdb.ps1 -Trigger $Trigger -Credential
$credent -ScheduledJobOption $elevat
```

Information about all included tasks in the Microsoft Windows Powershell:

```
Get-ScheduledTask -TaskPath \Microsoft\Windows\Pow*|
? state -ne Disabled
```

To get information about a specific task:

```
Get-ScheduledTask BackupDBTask| Get-ScheduledTaskInfo
```

To disable a specific task:

```
Get-ScheduledTask BackupDBTask| Disable-ScheduledTask
```

Task Scheduler allows you to run certain tasks, scripts at a specific time, without user intervention. Consider using custom applications to communicate with Task Scheduler using API calls.

1. Create a task “My Task” which runs the RunMe.bat script every day at 9 a.m.

   ```
   SchTasks /Create /SC DAILY /TN "My Task" /TR "C:RunMe.bat"
   /ST 09:00
   ```

2. Change “My Task” to run the script at 2 p.m.

   ```
   SchTasks /Change /TN "My Task" /ST 14:00
   ```

3. Create a task “My Task” which runs the RunMe.bat script on the first day of each month.

   ```
   SchTasks /Create /SC MONTHLY /D 1 /TN
   "My Task" /TR "C:RunMe.bat" /ST 14:00
   ```

4. Create a task “My Task” which runs the RunMe.bat script every working day at 2 p.m.

   ```
   SchTasks /Create /SC WEEKLY /D MON,TUE,WED,THU,FRI /TN "My Task"
   /TR "C:RunMe.bat" /ST 14:00
   ```

5. Removing a task named “My Task”

   ```
   SchTasks /Delete /TN "My Task"
   ```

Bulk task creation:

```
SchTasks /Create /SC DAILY /TN "Backup Data" /TR "C:Backup.bat"
/ST 07:00
SchTasks /Create /SC WEEKLY /D MON /TN "Generate TPS Reports"
/TR "C:GenerateTPS.bat" /ST 09:00
SchTasks /Create /SC MONTHLY /D 1 /TN "Sync Database" /TR "C:SyncDB.bat"
/ST 05:00
```

As with any other command line tool, you can include several commands in a batch file (script) to create (or delete) several tasks.