

CHROMOSOME in 120 Minutes



Fortiss GmbH
An-Institut der
Technischen Universität München
Guerickestr. 25
80805 München
chromosome@fortiss.org

Version 0.2
April 2, 2012

Copyright (c) 2011-2012, fortiss GmbH.
Licensed under the Apache License, Version 2.0.

Use, modification and distribution are subject to the terms specified in the accompanying license file LICENSE.txt located at the root directory of this software distribution. A copy is available at <http://chromosome.fortiss.org/>.

Contents

1	Introduction	3
1.1	The Name CHROMOSOME	3
1.2	Questions and Contact Information	4
1.3	Acknowledgements	4
2	Prerequisites (20 minutes)	5
3	CHROMOSOME in a Nutshell (30 minutes)	6
3.1	Goals of CHROMOSOME	6
3.2	Core Components	8
3.2.1	Node Manager	9
3.2.2	Login Server	9
3.2.3	Directory	9
3.2.4	Routing Table	9
3.2.5	Broker	9
3.2.6	Execution Manager	9
3.2.7	Resource Manager	9
3.2.8	Interface Manager	10
3.2.9	Primitive Components	10
3.2.10	Hardware Abstraction Layer	10
3.2.11	Advanced Components	10
3.2.12	Login Client Proxy and IP Login Server Proxy	10
3.3	Component Definition and Lifecycle	10
3.4	Component Instantiation	11
3.5	Sending and Receiving Data	12
3.6	Working with Tasks	13
4	Example 1: Hello World! (20 minutes)	14
5	Example 2: Chat with Chat Rooms (30 minutes)	20
6	Example 3: Chat Calculator (20+ minutes)	22
7	More Examples	29
A	Frequently Asked Questions	30
A.1	How can I create a new component?	30
A.2	How can I define a new topic?	30
A.3	How can I port CHROMOSOME to my own target platform?	30
B	Installing Visual C++ 2010 Express	32
C	Installing CMake	34
D	CHROMOSOME License	35

1 Introduction

For a long time, the focus of embedded systems development has been the implementation of isolated systems with clearly defined borders and interfaces. Recently, the trend of integrating these complex independent systems into larger *systems of systems* arises: manufacturing plants get connected with logistics, intelligent cars communicate with each other and the infrastructure. Due to the different life cycles of the different involved systems, adaptability in the sense of plug&play becomes more and more important. Systems must be integrated with other systems although the concrete systems were not known at design time. Future embedded systems have to be developed in a way so that they can be integrated into these systems of systems without losing their safety, security and real-time capabilities.

In order to achieve this, a powerful domain-independent software platform is required, which can flexibly be adapted to various application scenarios. CHROMOSOME¹ is a middleware and runtime system intended to meet these requirements. It combines features known from the embedded domain such as determinism where necessary with adaptivity known from internet technologies. CHROMOSOME treats extra-functional requirements as first-class entity and provides according mechanisms to fulfill the application requirements.

CHROMOSOME has a large set of designated features and is designed to evolve over time. It is completely open source and hence transparent to developers and end users. The current release includes a first subset of the features that will be available in future versions. Although we have a clear vision of the final system, CHROMOSOME's development is demand driven. Its goal is to provide adequate platform support and tooling for maximum efficiency.

The following tutorial will introduce you to the main concepts of CHROMOSOME and illustrate them with examples that are easy to understand. Including installation of required prerequisites, this tutorial will take about two hours to complete.

1.1 The Name CHROMOSOME

CHROMOSOME stands for Cross-domain Modular Operating System or Middleware. The name expresses the vision of CHROMOSOME:

1. We believe that in the future cross-domain solutions will be required.
2. As different applications might have many different requirements and the middleware will be deployed to very heterogeneous platforms, a scalable and modular solution is required.
3. CHROMOSOME will sometimes operate on top of an operating system (OS), but it might also replace an OS due to resource reasons. Therefore, we think that in the future, the boundary between an operating system and a middleware might get blurred.

Similar to biology, a CHROMOSOME instance can be built up a number of genes (modules). It is not the intention of CHROMOSOME to invent new genes, but instead to use existing protocols / solutions to implement components for specific tasks. The major idea is that CHROMOSOME offers a flexible blueprint that enables the selection of different solutions to adapt a system to the specific requirements.

Analog to the evolution of mankind, we do not believe that the genes, but also the blueprint provided by CHROMOSOME are initially perfect. Instead we believe in constant improvement based on discussions about specific components. This is one of the reasons why we are offering CHROMOSOME as open-source software. In case you have any questions or suggestions for improvement, we are looking forward to your comments.

¹<http://chromosome.fortiss.org/>

1.2 Questions and Contact Information

If you have any questions with respect to this tutorial or want to report a bug, please check the Frequently Asked Questions in Appendix A. If your question is not answered, please do not hesitate to send an e-mail to chromosome@fortiss.org. Thank you!

1.3 Acknowledgements

This work is partially funded by the following research grants:

- German Federal Ministry of Economics and Technology (BMWi) research grant 01ME12009 (project RACE²)
- German Federal Ministry of Economics and Technology (BMWi) research grant 01MA11002 (project AutoPnP³)
- Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology grant programme 1330/686 (Vorlaufforschung Fortiss)

²<http://www.projekt-race.de/>

³<http://www.autopnp.com/>

2 Prerequisites (20 minutes)

CHROMOSOME has been developed with platform independence in mind, but for sake of simplicity, this tutorial uses a Windows-based development environment and a Windows-based target platform. While installing the proposed build environment, you might want to start reading Section 3 giving an overview on CHROMOSOME and its features.

Apart from the CHROMOSOME source archive, two tools are recommended for building CHROMOSOME applications on Windows.

- **Visual Studio C++** (Express, Professional, Premium or Ultimate, preferably 2008 or 2010 versions): Visual Studio is Microsoft's platform for multi-language development. The so-called "Express Edition" is available free for evaluation purposes⁴. Note that CHROMOSOME has not yet been tested with Visual Studio 2011, of which a beta version has been recently released. You only have to install the C/C++ compiler, Additional packages like Silverlight or SQL Server are not required. Consult Appendix B for details.
- **CMake** (at least version 2.8.5): CMake is a cross-platform Makefile generator and is used to manage the build system. Output of CMake are the build system configurations (e.g., a UNIX Makefile, a Microsoft Visual Studio Project, an Eclipse CDT project). CHROMOSOME provides a customized set of macros to deal with components, dependencies, executables and documentation. CMake can be downloaded for free⁵. CHROMOSOME currently requires at least CMake version 2.8.5. It is *not* necessary to add CMake to the system search path. Consult Appendix C for details.

⁴<http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express>

⁵<http://www.cmake.org/cmake/resources/software.html>

3 CHROMOSOME in a Nutshell (30 minutes)

CHROMOSOME (often abbreviated by “XME”) is a domain-independent, data-centric⁶ middleware for cyber-physical systems. From the point of view of an application component, CHROMOSOME abstracts from basic functionality that is traditionally found in operating systems and middlewares, like scheduling and communication. This section will give the reader a very high level overview of the most important CHROMOSOME features. For in-depth information and concrete specifications, the reader is kindly referred to the commented source code of CHROMOSOME.

3.1 Goals of CHROMOSOME

CHROMOSOME is designed to match the requirements of future cyber-physical systems. As a research prototype, the main intention is to serve as a basis for discussion how future middleware/operating system architectures may look like. The development is done in a demand-driven way. Depending on the requirements of projects where CHROMOSOME is applied, new features in CHROMOSOME will be introduced.

The current version is the first version of CHROMOSOME and therefore provides only very basic functionality. Further features will be introduced in the future. In the following, the main concepts of CHROMOSOME are introduced. In addition, we state whether these features are already implemented or planned for future versions. Our point of view is that future middleware will not operate on a specific class of devices, but will run on very heterogeneous platforms. The exaggerated goal statement could be to “develop a middleware serving a range from 8-bit controllers to cloud servers”.

Therefore, scalability and modularity of the middleware is of highest importance. It must be easy to remove or add components from/to the middleware. Similar to micro-kernel operating systems, the core of CHROMOSOME must be very small and only contain the absolutely necessary features. Since the boundaries of operating systems, middleware, and applications are vanishing, one unique mechanism must be supported to add components on all these levels.

Concept 1: Data-Centric Design A very successful concept to achieve scalability is message-orientation as for example demonstrated by the QNX⁷ neutrino micro-kernel. However, one major drawback of this concept is the necessity of requiring knowledge of both the sender and the receiver. In the context of systems-of-systems where the communication partners are not known at design time, message orientation cannot be applied. Therefore, CHROMOSOME is based on the very powerful concept of data-centric design. Components specify which data they consume and produce and based on this information, the middleware calculates the communication routes.

It is important to note that in contrast to prominent other middleware systems relying on data-centric design, CHROMOSOME does not broadcast the data, instead it calculates specific routes based on the requirements. This allows for a very efficient implementation. Furthermore, the routes are calculated only when components are added or removed. Based on the experience in embedded systems, we assume that reconfiguration takes places less often than the transmission of data items. Similar to concepts from multimedia protocols, our initial assumption is that data transmission has to happen within real-time, while reconfiguration is not time-critical or has only soft real-time requirements.

Data is grouped by so-called *topics*. A topic is a data type with a certain structure. Examples for topics are temperature or rotation speed. Topics are defined for a specific domain. This enables the exchange of data between different applications. To allow cross-domain communication, we plan to define translation matrixes between the topics of different domains.

⁶For details, please refer to http://www.omg.org/news/whitepapers/Intro_To_DDS.pdf

⁷<http://www.qnx.com/>

Concept 2: Meta-Data Support (not yet implemented) Exchange of data can only be automated if the requirements on data are specified explicitly. This includes information about the quality of data such as age, accuracy, confidence, and safety level. In the future, we plan to support respective annotations to data, so-called *meta-data*, to specify these requirements.

Using meta-data, application components describe their requirements respectively guarantees on data. This information can be used to select appropriate communication streams while still retaining independence between sender and receiver.

We are planning to support two scenarios:

- **Static meta-data:** Application components have fixed requirements and guarantees. Meta-data are used to calculate the appropriate communication routes at configuration time. For example, imagine a temperature sensor network for building automation. Depending on its size, multiple sensors may be present in a room. Each sensor would specify its *location* as meta-data. A climate control component that controls the air conditioning system for a specific room would specify, using meta-data, that it only wants to receive temperature data from sensors that match the respective room.
- **Dynamic meta-data:** Application components producing data may specify meta-data such as the quality of the transmitted values dynamically during runtime. Consuming components may have dynamic meta-data acceptance filters and may use meta-data also for their calculations. For example, imagine that one of the rooms in the automated building from the previous example is a server room. In this case, a high reliability is required. Using meta-data about the confidence of each sensor, the climate control may dynamically calculate the confidence in its own set value for the air conditioning system. The association of data from the respective room could still use static meta-data.

Concept 3: Data Access Components can access data in two different ways. As standard operation, CHROMOSOME provides a publish-subscribe mechanism. Each component states which data is required and which data is produced. The middleware ensures the routing of the specific data. If data is only required seldomly in comparison to the data production rate, a request-response mechanism is offered to the application developer.

Concept 4: Timing (not yet implemented) Correct timing is a major concern for cyber-physical systems. CHROMOSOME abstracts the concrete implementation, but will offer mechanisms to specify end-to-end-timing and jitter requirements for data paths. The reason to abstract the concrete implementation / concrete configuration in CHROMOSOME is on the one hand to reduce the development effort by automatically deriving a concrete configuration satisfying the timing requirements (if a configuration exists) and on the other hand to support plug & play capability. The state of the art in embedded system design requires the developer to configure the application in a correct way. Several configurations might be valid, but typically only one configuration is specified. It is not possible for the run-time system to change the configuration, as the real requirements are not present any more and the system can not calculate alternative configurations. By explicitly stating the requirements and leaving configuration issues to the run-time system, this problem can be avoided. The system can support new components by calculating a new configuration that satisfies both the requirements of already running applications and the newly installed applications.

It is important to note that CHROMOSOME can of course only offer guarantees that are enabled by the underlying platform. In case CHROMOSOME runs on Windows, CHROMOSOME will not be able to satisfy jitter requirements in the range of microseconds. Nevertheless, different implementations of components will offer very good guarantees. The major concept here is to use algorithms that offer determinism sacrificing average-case performance. Examples are the implementation of a time-triggered communication scheme on top of Ethernet. This

implementation guarantees collision free communication, but comes with a lower bandwidth and flexibility.

Concept 5: Resource-Awareness (not yet implemented) CHROMOSOME is designed to be able to handle all resources of the platform. The major idea is that components / applications can state their worst-case resource requirements on memory, computation time, bandwidth, etc. CHROMOSOME will then ensure that enough resources are available to allow the execution of this component / application.

Concept 6: Other extra-functional properties – Fault-tolerance, QoS, Security (not yet implemented) CHROMOSOME is designed in a way such that components which guarantee other extra-functional properties can be easily extended. Examples on extra-functional properties that are currently being integrated in CHROMOSOME are fault-tolerance, QoS, and security.

Concept 7: Support for States (not yet implemented) A very important topic to ensure resource efficiency is state awareness. We are currently thinking about possibilities to support states (each one with a set of active applications) in CHROMOSOME. The support of (sub-)system wide states would simplify the development of applications, but also other aspects such as fault-tolerance.

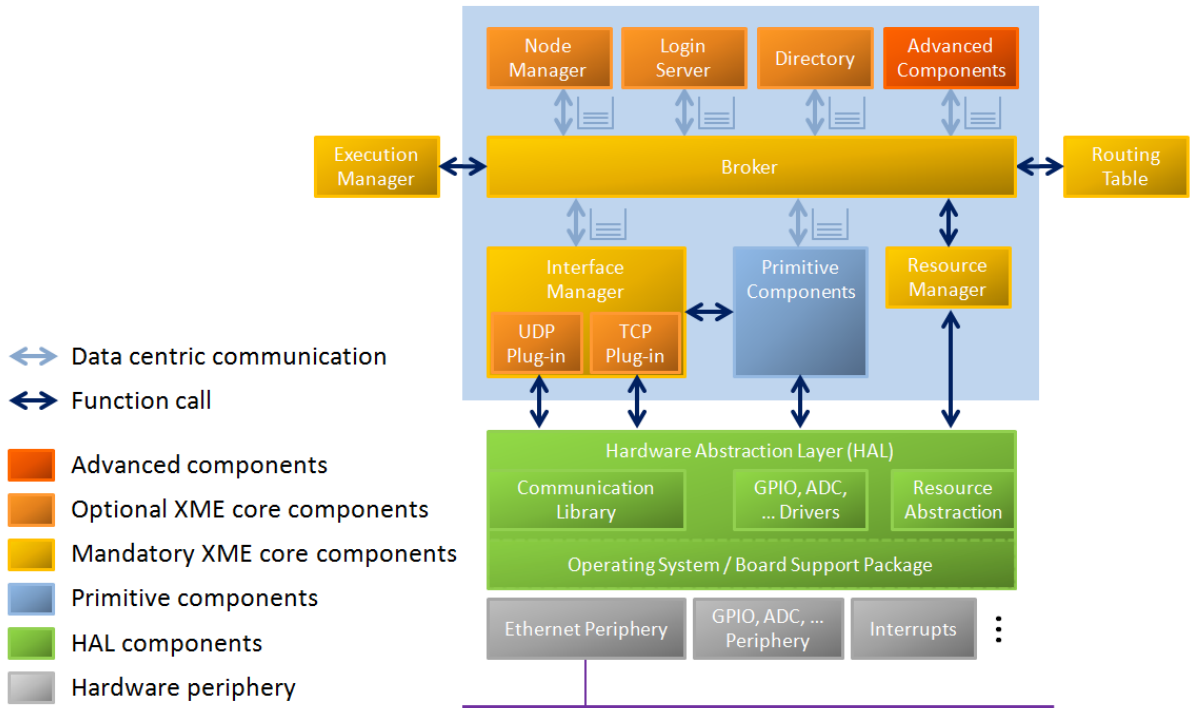


Figure 1: CHROMOSOME architecture overview.

3.2 Core Components

CHROMOSOME is designed in a modular fashion. Its component-oriented architecture is depicted in Figure 1. A (distributed) *application* consists of a set of (software) *components* that interact with each other by exchanging data. Each component is placed on a specific *node* in the network. On operating system based platforms like Windows, the software of a node is typically

represented by a console application. On embedded systems, the software of a node corresponds to the firmware. The following sections provide brief descriptions of the most important core components of CHROMOSOME. See Figure 1 for a graphical illustration.

3.2.1 Node Manager

A node can have an arbitrary amount of software components that implement the application. The *Node Manager* is responsible for connecting a new node to the network and obtaining a so-called *node identifier*. The node identifier is a unique address of the node within a network. The Node Manager periodically tries to obtain such a node identifier from the *Login Server*, which is present on one specific node in the network.

3.2.2 Login Server

The *Login Server* maintains a list of all nodes in the CHROMOSOME network. It receives login requests from the Node Managers of all other nodes and handles them accordingly. Only one Login Server should exist in the network.

3.2.3 Directory

The *Directory* maintains a list (a directory, as the name implies) of all publication and subscription requests in the network and is responsible for route calculation and setup. Usually, only one so-called *Master Directory* exists in the network, which is the authority for route calculation. All other nodes have a so-called *Local Directory*, which is only authoritative for local routes and forwards announcements (publication and subscription requests) to the Master Directory. Setup of routes is performed by modifying the *Routing Tables* present on every node.

3.2.4 Routing Table

As the name implies, the *Routing Table* is a data structure that contains routing information. In CHROMOSOME, routing is performed via so-called *data channels*. A data channel is a system-internal identifier for a communication path. When data arrive at a node, the Routing Table is inspected to determine where the data has to be delivered to. This is done by the *Broker*.

3.2.5 Broker

The *Broker* is responsible for delivering incoming data to the correct destination, which can be determined from the Routing Table. The Broker is also responsible for buffering the data if it can not be delivered immediately. The decision of when data can be delivered to a component is issued by the *Execution Manager*.

3.2.6 Execution Manager

The *Execution Manager* is the facility that enforces the model of execution in the system (according to a schedule). It determines when tasks are executed and when incoming data are processed by components. Data that are not immediately processed are put into queues.

3.2.7 Resource Manager

The *Resource Manager* is the node-local authority for resource reservation and arbitration. Among others, managed resources include memory and CPU time. Components report their requirements to the directory and only get them assigned when enough resources are available.

3.2.8 Interface Manager

The *Interface Manager* abstracts from the communication interfaces available at a certain node. Typical submodules of the Interface Manager are facilities for communication over Ethernet/IP with UDP or TCP. When data is to be sent, it is delivered from the Broker to the Interface Manager. Incoming data from the network is forwarded to the Broker to decide where to deliver it to.

3.2.9 Primitive Components

Primitive Components directly access the *Hardware Abstraction Layer* using function calls and offer data obtained from the hardware in a data centric way or forward data to the respective actuators. They are typically the source of physical data obtained from the environment and the sink for control data to the environment.

3.2.10 Hardware Abstraction Layer

The *Hardware Abstraction Layer*, often abbreviated by HAL, provides a platform independent application programming interface (API). CHROMOSOME components are implemented against that API and hence are platform independent. The HAL typically implements hardware-specific functionality such as drivers for microcontroller periphery like general purpose I/O (GPIO) or analog to digital conversion (ADC).

3.2.11 Advanced Components

The actual (distributed) application running in the network is implemented by application-specific *Advanced Components*. Advanced Components operate solely on data obtained via data centric communication. In particular, no direct access to the Hardware Abstraction Layer with effects in the environment is allowed.

3.2.12 Login Client Proxy and IP Login Server Proxy

These components are required for new nodes to initially contact the Login Server. A node that does not have a Login Server will usually have a *Login Server Proxy* component for each of its physical network interfaces. These components are named according to their associated interface types, for example *IP Login Server Proxy* for an Ethernet interface with IP communication stack. Whenever the Node Manager, which is responsible for periodic login requests, issues a login request, the Login Server Proxy takes care of broadcasting it over its associated communication interface.

Nodes that have a Login Server component usually also have a *Login Client Proxy* for each of their interfaces. The Login Client Proxy will receive the login requests issued by the Login Server Proxy on the respective medium and forward them locally to the Login Server. The login response is forwarded in a similar manner. This concept makes it possible to separate the login and address assignment logic from specific interface types and communication protocols and enables a fine-grained assignment of functionality to each communication interface.

3.3 Component Definition and Lifecycle

CHROMOSOME components are assigned to one of the following categories according to their “layer” within the system: **adv** (advanced), **prim** (primitive), **core** (runtime system) or **hal** (hardware abstraction). A CHROMOSOME component is defined by four functions to *create*, *activate*, *deactivate* and *destroy* it, illustrated here for a component called `xme_adv_myComponent`:

```

xme_adv_myComponent_create(xme_adv_myComponent_configStruct_t*);
xme_adv_myComponent_activate(xme_adv_myComponent_configStruct_t*);
xme_adv_myComponent_deactivate(xme_adv_myComponent_configStruct_t*);
xme_adv_myComponent_destroy(xme_adv_myComponent_configStruct_t*);

```

The distinction between creation/activation and deactivation/destruction is required in case a component needs to be migrated: in this case, the component is first deactivated (to obtain a consistent state), then moved and activated at the new location. All four functions take a pointer to a configuration structure, which represents the state of the respective component instance. Multiple component instances of the same component type usually have their own configuration structures.

A typical task during creation of a component is registration of data demand and data production. Furthermore, a component may create (periodic) tasks to implement its functionality. These two concepts are illustrated in the following sections.

3.4 Component Instantiation

Components can be instantiated by adding them to the so-called *component list*, usually defined in the main program file. Mandatory core components (compare Figure 1) are implicitly added to the list. An initial configuration may be provided for each component. Listing 1 shows a sample component list with respective configurations.

Listing 1: Sample component list with component configurations.

```

1  /*****
2  /***   Component configurations                               ***/
3  *****/
4  XME_COMPONENT_CONFIG_INSTANCE(xme_core_nodeManager) =
5  {
6      0x00000000021041A1, // deviceType
7      XME_CORE_DEVICE_GUID_RANDOM // deviceGuid
8  };
9
10 XME_COMPONENT_CONFIG_INSTANCE(xme_prim_ipLoginServerProxy, 1);
11
12 /*****
13 /***   Component descriptor                               ***/
14 *****/
15 XME_COMPONENT_LIST_BEGIN
16     XME_COMPONENT_LIST_ITEM(xme_core_nodeManager, 0)
17     XME_COMPONENT_LIST_ITEM(xme_prim_ipLoginServerProxy, 0)
18     XME_COMPONENT_LIST_ITEM_NO_CONFIG(xme_adv_myComponent)
19 XME_COMPONENT_LIST_END;

```

This example declares that three components (besides internal core components), namely `xme_core_nodeManager`, `xme_prim_ipLoginServerProxy` and `xme_adv_myComponent`, will be present on this node. We can distinguish between the following types of components:

1. **Components with internal state:** The `XME_COMPONENT_LIST_ITEM()` macro is used to declare such a component (compare lines 16 and 17). The second parameter of the macro is the zero-based index of the respective configuration (defined above the list) that is used to initialize the component.

Some components expect some of their configuration variables to be initialized properly. For example, the *Node Manager* expects its *device type* and *device GUID* members to be set up. This is achieved by assigning values to the respective members when declaring the configuration structure with the `XME_COMPONENT_CONFIG_INSTANCE` macro (compare

lines 4–8). Configurations for multiple components of the same type can be declared by separating multiple initialization blocks with commas and using the respective index in the `XME_COMPONENT_LIST_ITEM()` macro.

Other components, like the *IP Login Server Proxy*, do not require any special configuration. They only need a properly defined configuration structure to store their state during runtime. Again, the `XME_COMPONENT_CONFIG_INSTANCE` macro is used with the second parameter set to the number of configuration structures for that component type to create (we could also declare multiple components of the same type), in this case one (line 10).

2. **Components without internal state:** The `XME_COMPONENT_LIST_ITEM_NO_CONFIG()` macro is used to declare such a component (compare line 18). In this case, `NULL` will be passed in the `config` parameters of the four functions introduced in Section 3.3.

The list item macro calls have to be embedded between the `XME_COMPONENT_LIST_BEGIN` and `XME_COMPONENT_LIST_END` macro calls. Components declared in this list are automatically created and activated during startup and deactivated and destroyed during shutdown.

3.5 Sending and Receiving Data

Before sending data, a component states its intent to the runtime system. This is necessary to allow the appropriate communication routes to be set up and is usually performed from within a component's *create* function. For using data centric communication, `#include` the file `"xme/core/dcc.h"`. To inform CHROMOSOME that a component intends to send data under a specific topic, call:

```
publicationHandle =
    xme_core_dcc_publishTopic(
        topic, XME_CORE_MD_EMPTY_META_DATA, NULL
    );
```

The actual sending of data is performed with:

```
xme_core_dcc_sendTopicData(
    publicationHandle, &data, sizeof(data)
);
```

A component can subscribe to a certain topic with a given `receiveTopicCallback` function by calling:

```
subscriptionHandle =
    xme_core_dcc_subscribeTopic(
        topic, XME_CORE_MD_EMPTY_META_DATA, receiveTopicCallback, NULL
    );
```

Whenever data matching the given topic is received, the given callback function is invoked. The last parameter of the function can be used to pass user-defined data to the callback function, for example a pointer to the component's configuration structure. `receiveTopicCallback` must have a signature matching `xme_core_dcc_receiveTopicCallback_t`, namely:

```
void receiveTopicCallback(xme_hal_sharedPtr_t dataHandle, void* userData);
```

`dataHandle` is a reference to the memory where the received data is located.

3.6 Working with Tasks

Tasks are allocated via the resource manager, hence `#include "xme/core/resourceManager.h"` to use them. A component can create one or multiple asynchronous tasks by calling

```
taskHandle =
    xme_core_resourceManager_scheduleTask(
        startMs, periodMs, XME_HAL_SCHED_PRIORITY_NORMAL,
        taskCallback, NULL
    );
```

This function registers the `taskCallback` function to be scheduled for execution according to `startMs` (number of milliseconds before first invocation) and `periodMs` (interval between invocations). The last parameter of the function can be used to pass user-defined data to the callback function, for example a pointer to the component's configuration structure. `taskCallback` must have a signature matching `xme_hal_sched_taskCallback_t`, namely:

```
void taskCallback(void* userData);
```

A task can be suspended or resumed by calling:

```
xme_hal_sched_setTaskExecutionState(
    taskHandle, <flag>
);
```

`<flag>` is a Boolean flag that indicates the new task execution state. The call will block until the new state can be enforced, which is not the case while the task's callback function is executed. A task can be aborted by calling:

```
xme_core_resourceManager_killTask(
    taskHandle
);
```

If this function is called from the task to kill itself, then it will mark the task for deletion and return immediately. If it is called from a different task, the function blocks until the task has been removed. Similarly to suspension and resuming, a task is not removed until its callback function has returned.

4 Example 1: Hello World! (20 minutes)

Two steps are required to build an application based on CHROMOSOME: the build system has to be generated using CMake and the application needs to be compiled to binary code.

After the installation of CMake and Visual Studio, the following steps are required:

1. Download the Windows release archive of CHROMOSOME from the website.⁸
2. Extract the archive to a directory of your choice (compare Figure 2).⁹ After extraction, the subdirectory `src` contains the source code for CHROMOSOME. From now on, we will call that directory `<XME_ROOT>`. The subdirectory `bin` contains binaries shipped with the release. This allows testing of CHROMOSOME without compiling your own applications.

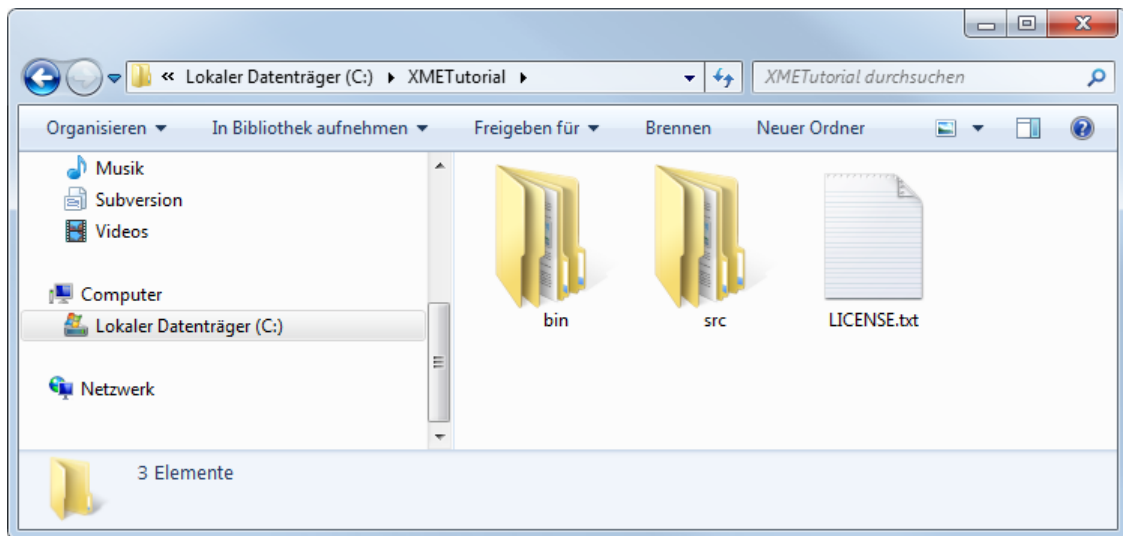


Figure 2: Files and folders extracted from the source archive.

3. Use the *start menu* to run CMake (*cmake-gui*) (compare Figure 3).
4. In the *Where is the source code* field, select the full path to the directory `<XME_ROOT>/examples/tutorial` (compare Figure 4).

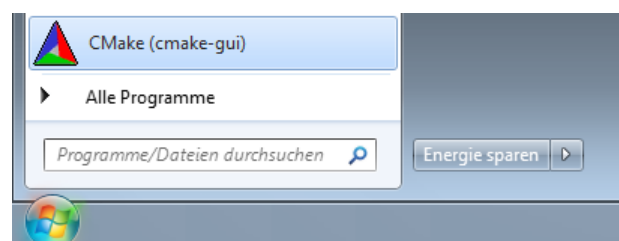


Figure 3: CMake (*cmake-gui*) icon in the start menu.

5. In the *Where to build the binaries* field, select the full path to the directory `<XME_ROOT>/examples/build/tutorial` (note the additional `build` folder, this folder does not yet exist).
6. Click the *Configure* button.

⁸CHROMOSOME: <http://chromosome.fortiss.org/>

⁹Please ensure that the path name is not extraordinary long, as this could cause issues with the build system.

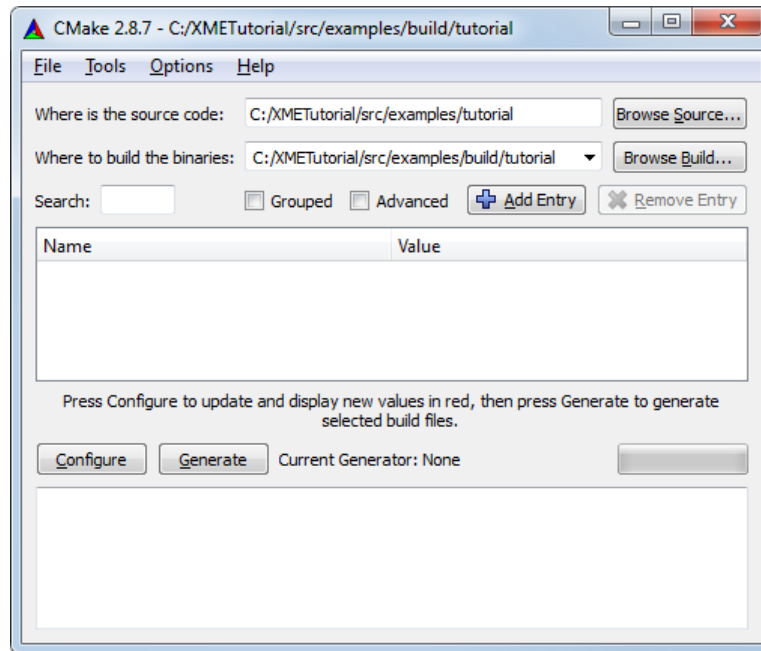


Figure 4: Source code and build directory specification.

7. Choose the *Visual Studio* toolchain that corresponds to your Visual Studio version (do *not* use the 64 bit version) and click *Finish* (compare Figure 5).

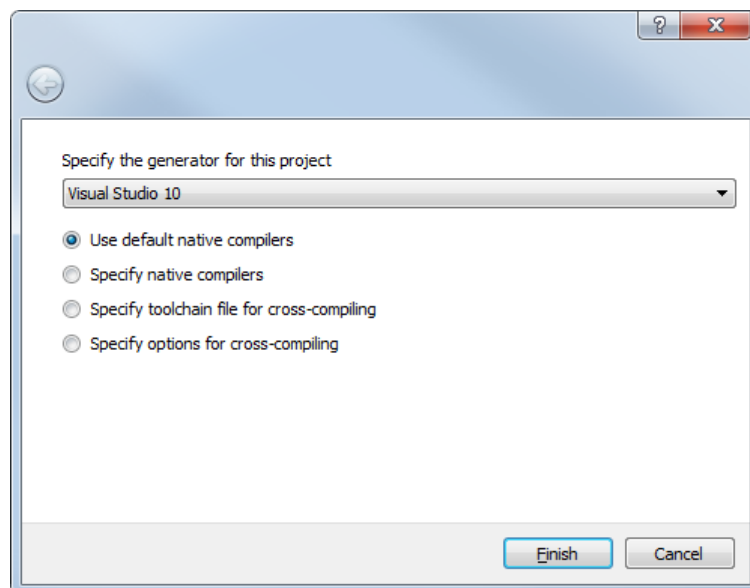


Figure 5: Toolchain selection in CMake.

8. After CMake has finished its configuration, various configuration variables marked in red should appear in the list (compare Figure 6).
9. Click the *Generate* button. This will generate the Visual Studio solution file `Tutorial.sln` which you can open in Visual Studio. The file will be placed in the following directory: `<XME_ROOT>/examples/build/tutorial` (compare Figure 7).

To compile the exemplified application, the following steps are required:

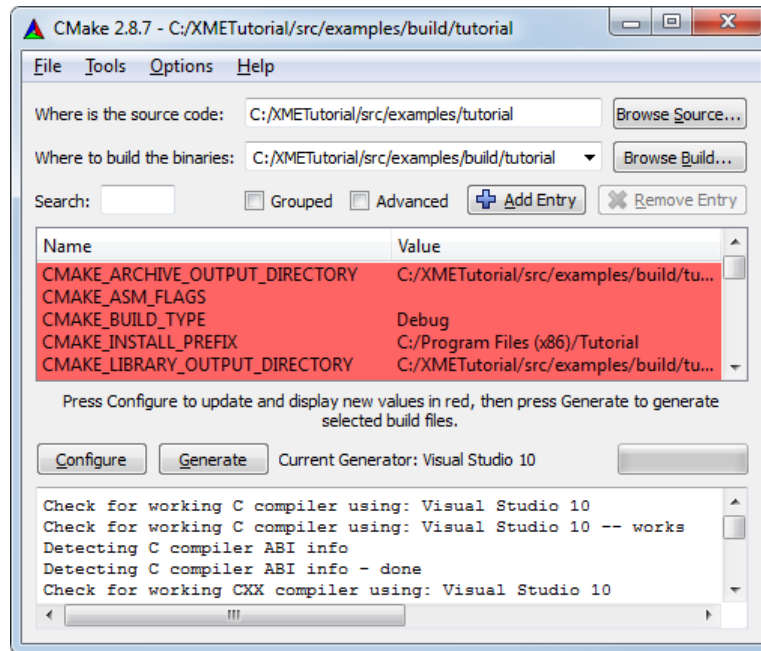


Figure 6: Configuration and build system generation in CMake.

1. Fire up Visual Studio and select *File* → *Open* → *Project/Solution...*
2. Navigate to the `<XME_ROOT>/examples/build/tutorial` directory and select the solution file `Tutorial.sln`.
3. After loading the solution, you will see a project tree in the left-hand pane. Right-click on the `tutorial` project and choose *Set as StartUp Project* (compare Figure 8).
4. In the tool bar, select the solution configuration you want to build (usually **Debug** or **Release**). The **Debug** build includes debugging information and should be used for development. If in doubt, choose **Release**.
5. Hit *F7* to compile the solution.
6. Debug/run the project as usual (e.g., hit *F5* to debug or *Ctrl+F5* to run without debugging). If you get prompted whether to rebuild the out-of-date `ZERO_CHECK` project (compare Figure 9), you may select the *Do not show this dialog box again* check box and choose *Yes*.
7. When the application starts, you will probably receive a query from the *Windows Firewall*. **CHROMOSOME** is designed to talk to other nodes in the network and hence needs a firewall exception for full functionality. If you do not want the **CHROMOSOME** to talk to other nodes, then simply deny access.¹⁰ In this case, communication will be limited to **CHROMOSOME** applications on the local computer. If you want **CHROMOSOME** applications to communicate with each other in the local network, allow access to the *Home or Company Network* as shown in Figure 10.

This tutorial is bundled with a simple *Hello World* example application, which gets built when this workflow is executed. The entry point into the executable is in `tutorial.c`, which also defines the **CHROMOSOME** components used in this application. The application is composed

¹⁰If you want to allow access to the network at a later point in time, you can change the Firewall settings in the *System Control Panel*.

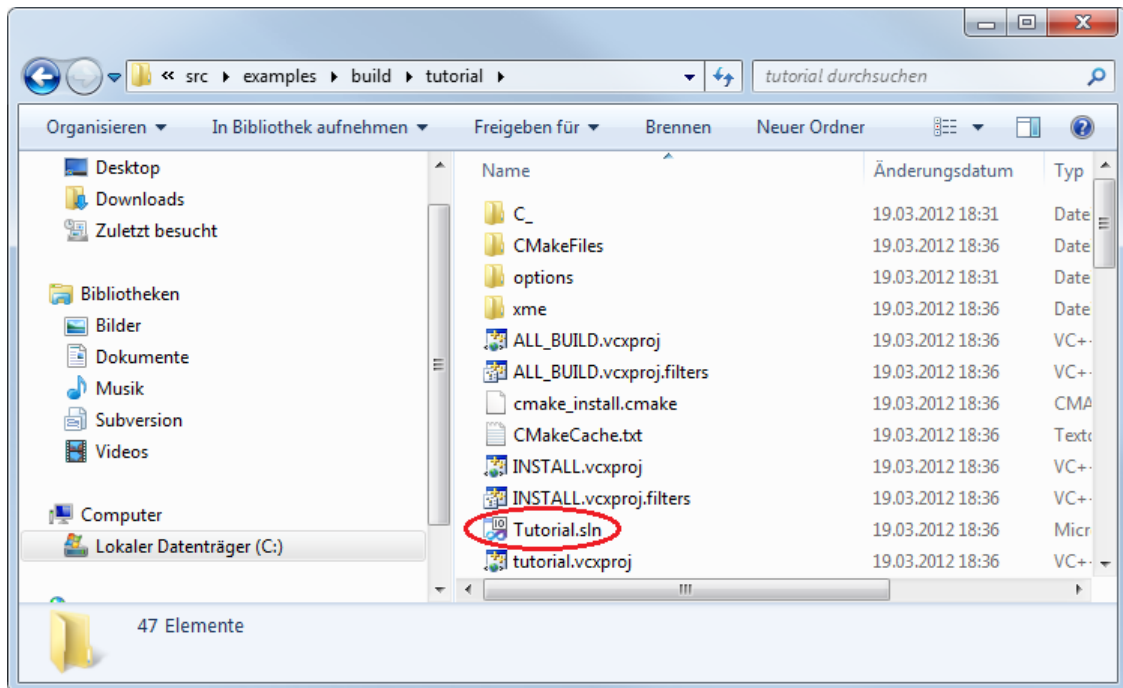


Figure 7: Build directory after build system generation, highlighted in red the Visual Studio solution file.

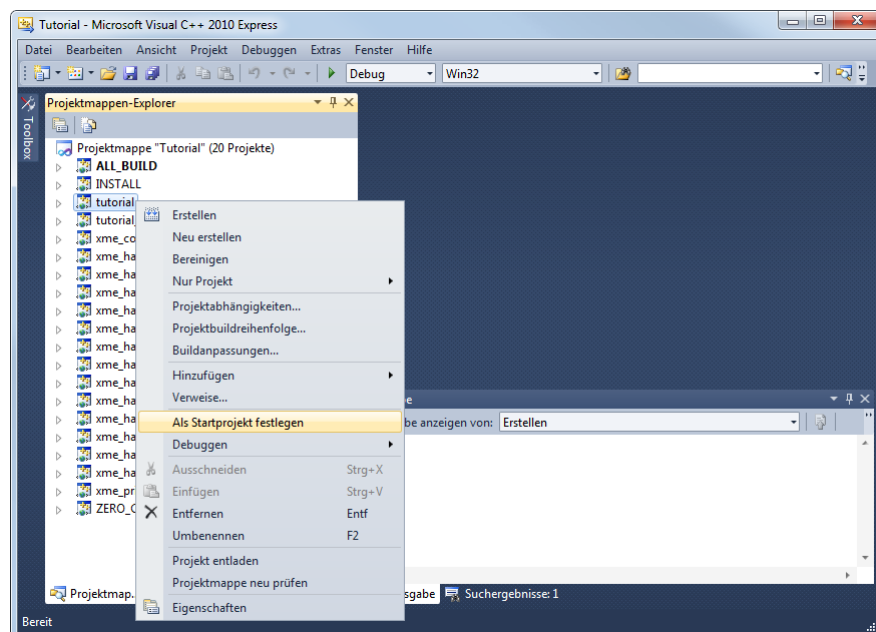


Figure 8: Setting the `tutorial` project as *StartUp Project*.

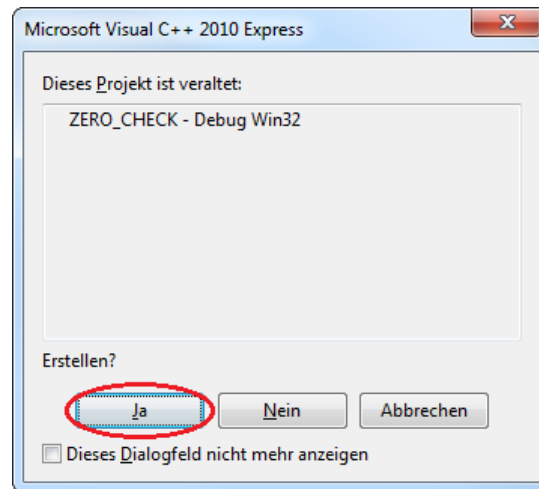


Figure 9: ZERO_CHECK project out of date prompt.

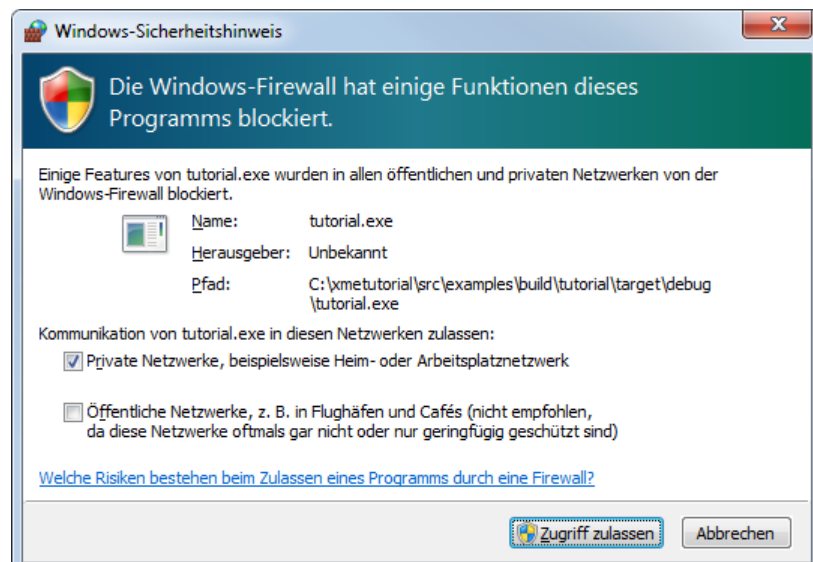


Figure 10: Firewall settings for the Tutorial application.

of some software components that are explained in section 3.2 (*Node Manager*, *IP Login Server Proxy*) and an application-specific components called *Hello World Component*). The latter one prints the text “Hello World!” to the console every two seconds (compare Figure 11). You may now inspect the source code in the files `tutorial.c`, `helloWorldComponent.c` and `helloWorldComponent.h` to see how the concepts from Section 3 are used in the application.

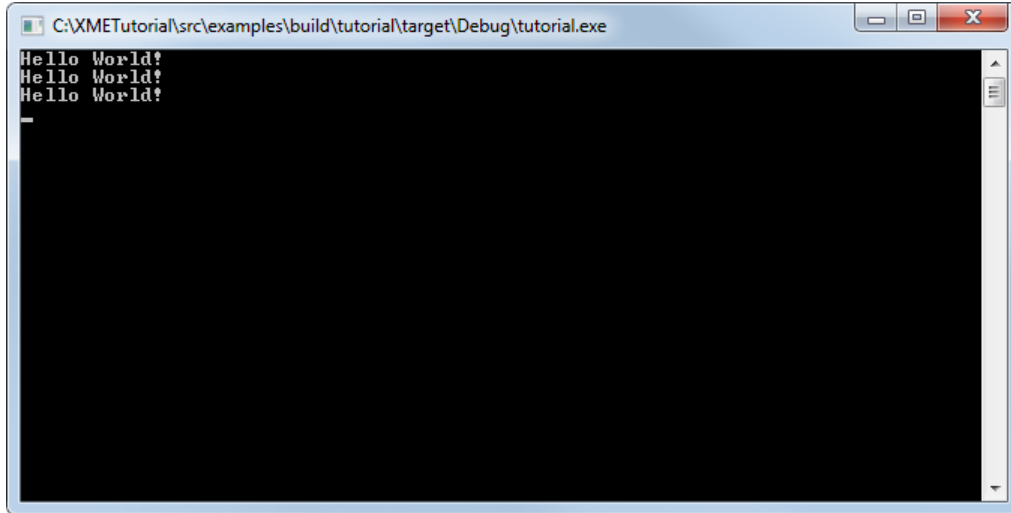


Figure 11: *Hello World Component* printing messages to the console.

5 Example 2: Chat with Chat Rooms (30 minutes)

It is now time to replace the *Hello World* application with a distributed *Chat* application. This tutorial comes with components that realize a distributed chat service. The source code of this application is already present in the tutorial project, we just need to configure it properly.

1. The *Hello World* application was using a specific component to print its “Hello World!” message. As this message is quite annoying in a chat room, we first have to remove the *Hello World* component from the tutorial project. At the same time, we enable the *Chat Component* which implements the chat client. This has to be done in the project’s main file `tutorial.c`, which can be opened in Visual Studio by double-clicking on the respective item below *tutorial* → *Source Files* in the *Project Explorer* pane. Please comment the line with the `helloWorldComponent` and uncommend (enable) the line with the `chatComponent` as shown below:

```

48 /*****
49 /***   Component descriptor                               ***/
50 /*****
51 XME_COMPONENT_LIST_BEGIN
52     XME_COMPONENT_LIST_ITEM(xme_core_nodeManager , 0)
53     XME_COMPONENT_LIST_ITEM(xme_prim_ipLoginServerProxy , 0)
54 /* XME_COMPONENT_LIST_ITEM(helloWorldComponent, 0)*/ // Comment this line
55     XME_COMPONENT_LIST_ITEM(chatComponent , 0)        // Uncomment this line
56 XME_COMPONENT_LIST_END;
```

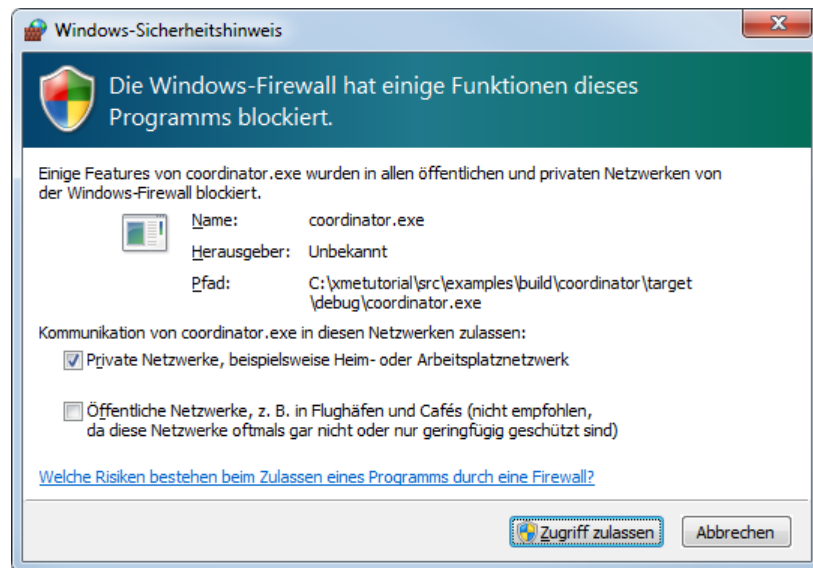
2. The routing of messages is performed automatically by CHROMOSOME core components. To keep the tutorial project small, all required components for network management and routing messages between distributed nodes (i.e., *Login Server* and *Master Directory*) have been added to a separate project named **Coordinator**. We will treat the Coordinator as a separate node and have it running in the background so that people can dynamically enter chat rooms.

To be able to use this functionality, please go go back to Section 4 and perform the described steps again for the `coordinator` example. Please use the directory `<XME_ROOT>/examples/coordinator` as input for the *Where is the source code* field and use the directory `<XME_ROOT>/examples/build/coordinator` as input for the *Where to build the binaries* field within CMake. After generation of the build system, the `Coordinator.sln` Visual Studio solution file can be found in at `<XME_ROOT>/examples/build/coordinator`. Do not forget to set up `coordinator` as the *Startup Project* within Visual Studio.

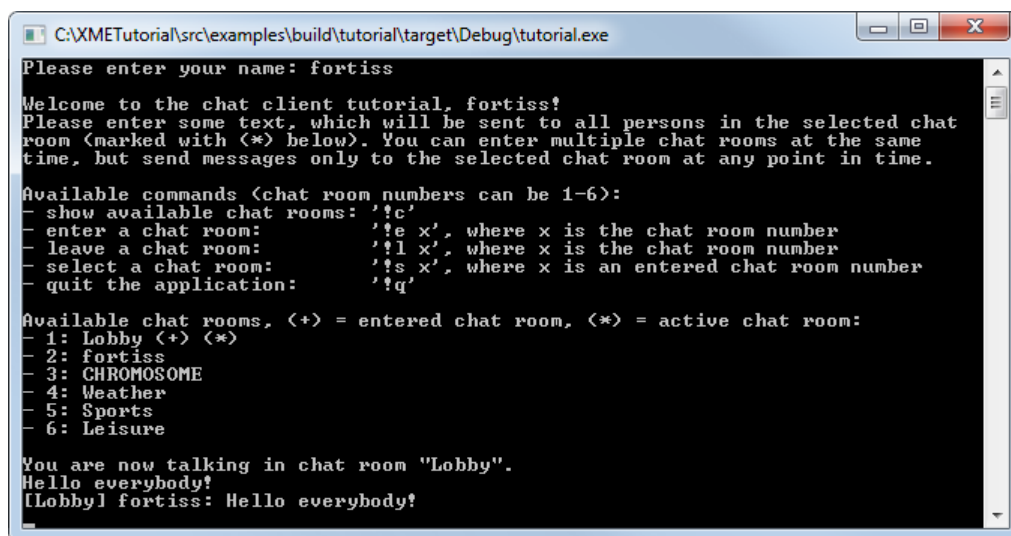
3. After having successfully compiled the coordinator project as well as the tutorial project, a basic distributed chat application can be started. In order to do this, execute **one instance** of the coordinator application. If you receive a request from the *Windows Firewall*, please allow communication to the *Home or Company Network* as shown in Figure 12. Note that if you have colleagues on the same network running this tutorial at the same time, you should talk to them to only run one instance of the Coordinator at any point in time.¹¹

The Coordinator can either be started from Visual Studio or by executing `<XME_ROOT>/examples/build/coordinator/target/*/coordinator.exe`. This will open a terminal window in which CHROMOSOME may print logging information. By default, the log level is set up so that only warnings and errors will be shown, so it is normal for the terminal window to not show any activity.

¹¹This is a limitation of the current version of CHROMOSOME. We do not yet support multiple subnets or detect the presence of other Coordinator instances. This will change in future versions.

Figure 12: Firewall settings for the **Coordinator** application.

4. We can now execute an arbitrary number of instances of the tutorial application. To do this, navigate to the respective directory and launch `<XME_ROOT>/examples/build/tutorial/target/*/tutorial.exe`. This will also open a terminal window, which first prompts to enter a name. After a name is given, you will be presented a list of commands and chat rooms to choose from (compare Figure 13).

Figure 13: Sending chat messages via the *Chat Component*.

The semantics of a chat room are that messages sent to a chat room should only be visible to people that have joined the respective chat room. This concept can be directly mapped to the notion of *topics*.

Anything that is not a command is interpreted as a message to send to the *active* chat room. The active chat room can be changed with the `!s` command.

Launch multiple instances of the chat example using different names and send some messages. Then run the application in a distributed way using multiple computers on the same subnet.

6 Example 3: Chat Calculator (20+ minutes)

After having modified an already existing component in Section 5, it is now time to create a completely new component. A so-called *Chat Calculator* component is a good starting point for digging deeper into CHROMOSOME. The calculator should fulfill the following requirements:

- Join a chat room.
- Announce its presence in the chat room every 30 seconds.
- Listen to commands of the form “!calc <operand1> <operation> <operand2>”.
- Support addition, subtraction, multiplication and division as <operation>.
- Send calculated results to the chat room in the form “<operand1> <operation> <operand2> = <result>”.
- Gracefully handle division by zero.

The following steps guide you through the implementation of the *Calculator Component*. Please be warned that the given time frame of 20 minutes does not include thorough reading and understanding of the code. You can find electronic versions of the code for the *Calculator Component* on the CHROMOSOME website¹².

1. Use the *Template Generator* application to create a new component with name “Chat Calculator” of class adv (advanced). The *Template Generator* application can be found in the directory <XME_ROOT>/../bin/windows_x86 and is explained in FAQ A.1.

The application will generate two files named `chatCalculator.h` and `chatCalculator.c` in the directory <XME_ROOT>/xme/adv and fill it with sample content. Furthermore, it will automatically add the new component to the `CMakeLists.txt` file in the same directory. This is the quickest way to generate a new CHROMOSOME component.

2. Edit the `CMakeLists.txt` file of the <XME_ROOT>/examples/tutorial project to use the new component:

```

97 # Build XME components
98 xme_link_components(
99     "tutorial"
100     xme_prim_ipLoginServerProxy
101     xme_core_core
102     xme_hal_dio
103     xme_hal_net
104     xme_hal_sleep
105     xme_adv_chatCalculator    # Add this line
106 )

```

Note that CMake will automatically pick up the changes in `CMakeLists.txt` when we perform the next full compile. This is why you do not have to rerun CMake manually after this change.

3. Open the Tutorial solution with the modifications from Section 5 in *Visual Studio* and edit `tutorial.c` to include the new component:

```

25 /*****
26 /** Includes ****
27 /*****
28 #include "chatComponent.h"

```

¹²<http://chromosome.fortiss.org/>

```

29 #include "helloWorldComponent.h"
30 #include "xme/core/componentList.h"
31 #include "xme/prim/ipLoginServerProxy.h"
32 #include "xme/adv/chatCalculator.h" // Add this line

```

4. Declare the component configuration and add an instance of the component to the component list in `tutorial.c`. Apply the following two modifications:

```

48 /*****
49 /** Component configurations */
50 *****/
51 XME_COMPONENT_CONFIG_INSTANCE(xme_core_nodeManager) =
52 {
53     0x00000000021041A1, // deviceType
54     XME_CORE_DEVICE_GUID_RANDOM // deviceGuid
55 };
56
57 XME_COMPONENT_CONFIG_INSTANCE(xme_prim_ipLoginServerProxy, 1);
58
59 XME_COMPONENT_CONFIG_INSTANCE(helloWorldComponent, 1);
60
61 XME_COMPONENT_CONFIG_INSTANCE(chatComponent, 1);
62
63 XME_COMPONENT_CONFIG_INSTANCE(xme_adv_chatCalculator) = // Add this block
64 {
65     // public
66     CHAT_TOPIC_BASE // topic
67 };
68
69 /*****
70 /** Component descriptor */
71 *****/
72 XME_COMPONENT_LIST_BEGIN
73     XME_COMPONENT_LIST_ITEM(xme_core_nodeManager, 0)
74     XME_COMPONENT_LIST_ITEM(xme_prim_ipLoginServerProxy, 0)
75 // XME_COMPONENT_LIST_ITEM(helloWorldComponent, 0)
76     XME_COMPONENT_LIST_ITEM(chatComponent, 0)
77     XME_COMPONENT_LIST_ITEM(xme_adv_chatCalculator, 0) // Add this line
78 XME_COMPONENT_LIST_END;

```

5. After these changes, press *F7* in *Visual Studio* to build the solution. This is required for CMake to pick up the changes in the build system. You will probably notice CMake re-run automatically in the *Visual Studio* console. You might be prompted whether to reload the *Visual Studio Solution*, in which case you should choose *Reload* (compare Figure 14).

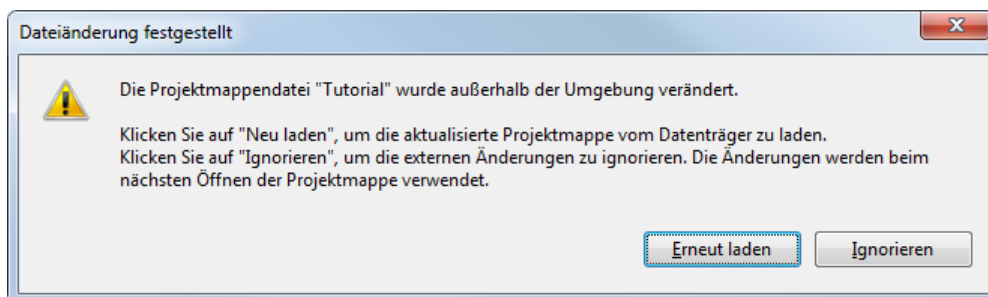


Figure 14: *Visual Studio Solution* reload prompt after *CMake* has regenerated the build system.

6. You should now see the `xme_adv_chatCalculator` item in the *Project Navigator* pane in *Visual Studio*. You can comfortably edit the source and header file of the component from there.

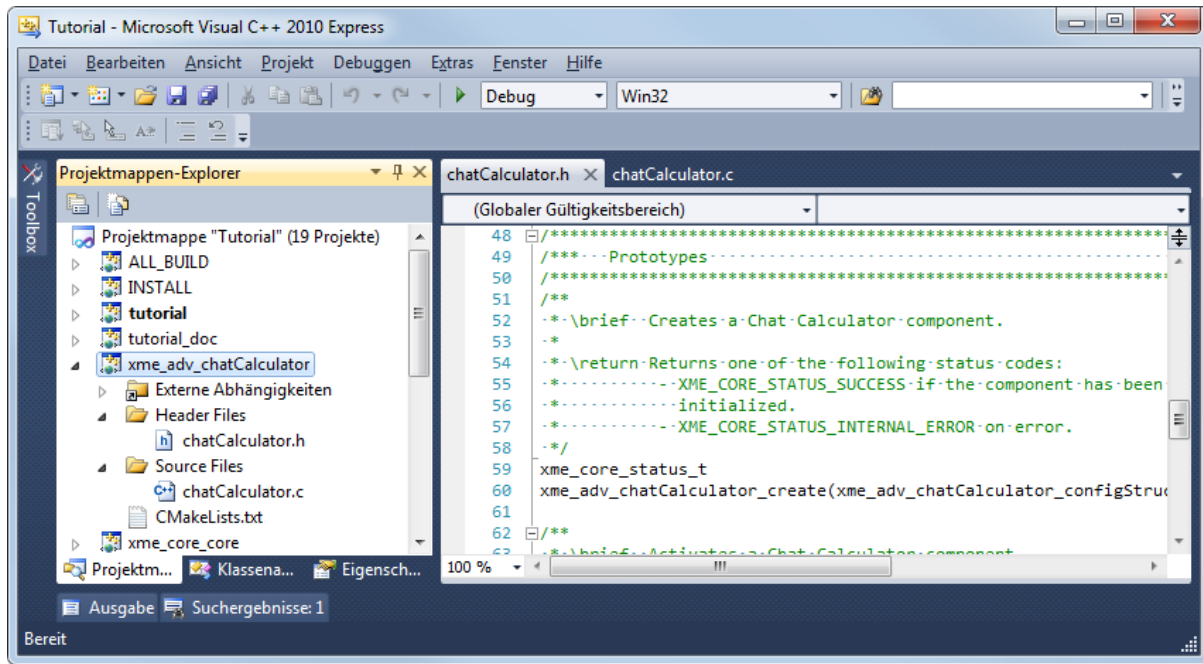


Figure 15: `xme_adv_chatCalculator` component in the *Visual Studio Project Navigator* pane.

7. The component we have just added comes with some example code that we have to adapt in order to implement the *Chat Calculator*. Open the respective files `chatCalculator.h` and `chatCalculator.c` found in `<XME_ROOT>/xme/adv` to make yourself familiar with the generated code. You will notice that the generated example code publishes and subscribes the `XME_CORE_TOPIC_EVENT` topic and that a task is created that prints a message every two seconds. The publication and subscription handles as well as the task handle is stored in the component's configuration structure `xme_adv_chatCalculator_configStruct_t`.

We can reuse the existing functionality for implementing the *Chat Calculator*, because the calculator has to listen (read: subscribe) to a specific chat room and print (read: publish) the result of the calculation. Furthermore, we need to announce the presence of the *Chat Calculator* every 30 seconds, for which the task can be used.

In order to ensure maximum flexibility (and to demonstrate how to use configuration variables), we will configure the topic of the chat room to enter (i.e., the topic) in *Chat Calculator*'s configuration. For this, we will define a member variable called `topic` in the configuration structure that we can set to an arbitrary value in our main program when we instantiate the component. In `chatCalculator.h`, apply the following two changes:

```

33  /*****
34  /***   Type definitions
35  *****/
36  typedef struct
37  {
38      // public
39      xme_core_topic_t topic;                // Change this line
40      // private
41      /* int dummyState; */                  // Comment or remove this line
42      xme_core_resourceManager_taskHandle_t taskHandle;
43      xme_core_dcc_publicationHandle_t publicationHandle;
44      xme_core_dcc_subscriptionHandle_t subscriptionHandle;
45  }
46  xme_adv_chatCalculator_configStruct_t;

```

We can leave the rest of the header file as-is.

8. Now it is time to implement the actual functionality in `chatCalculator.c`. Start by changing the published and subscribed topic to the chat room in which we want the *Chat Calculator* to be. Apply the following four changes:

```

69 xme_core_status_t
70 xme_adv_chatCalculator_create(xme_adv_chatCalculator_configStruct_t* config)
71 {
72     // TODO: Initialize component state
73     /* config->dummyState = 0; */ // Comment or remove this line
74
75     /* // TODO: Add code */ // Comment or remove this block
76     // XME_LOG(XME_LOG_NOTE, "Create function of Chat Calculator called!\n");
77
78     // Example: Publish a topic
79     config->publicationHandle =
80         xme_core_dcc_publishTopic
81         (
82             config->topic, // Change this line
83             XME_CORE_MD_EMPTY_META_DATA,
84             false,
85             NULL
86         );
87
88     // Check for errors
89     if (XME_CORE_DCC_INVALID_PUBLICATION_HANDLE ==
90         config->publicationHandle)
91     {
92         return XME_CORE_STATUS_INTERNAL_ERROR;
93     }
94
95     // Example: Subscribe to a topic
96     config->subscriptionHandle =
97         xme_core_dcc_subscribeTopic
98         (
99             config->topic, // Change this line
100             XME_CORE_MD_EMPTY_META_DATA,
101             false,
102             _xme_adv_chatCalculator_receiveDataCallback,
103             config
104         );
105
106     [...] // Leave the rest of the function as-is
107 }

```

9. Now we should adapt the `_xme_adv_chatCalculator_receiveDataCallback()` function, which handles incoming data for the subscribed topic:

```

28 /***** Implementation *****/
29 /** Implementation */
30 /*****
31 // TODO: Use or remove this sample topic subscription callback function:
32 static
33 void
34 _xme_adv_chatCalculator_receiveDataCallback(xme_hal_sharedPtr_t dataHandle,
35                                             void* userData)
36 {
37     // TODO: In case you supply the component's configuration instance
38     // via userData when subscribing, you can access it here:
39     xme_adv_chatCalculator_configStruct_t* config =
40         (xme_adv_chatCalculator_configStruct_t*)userData;
41
42     /* // TODO: Add code */ // Comment or remove this block
43     // XME_LOG(XME_LOG_NOTE, "Receive data callback function called!\n");

```

```

42
43 /* // Example: Print data to screen */      // Comment or remove this block
44 // {
45 //     const char* message = xme_hal_sharedPtr_getPointer(dataHandle);
46 //     XME_LOG(XME_LOG_NOTE, "Received: %s\n", message);
47 // }
48
49 // Add all the following lines
50
51 char* input = (char*)xme_hal_sharedPtr_getPointer(dataHandle);
52 int size = xme_hal_sharedPtr_getSize(dataHandle);
53
54 // Output data
55 char output[512];
56
57 // Sanitize input
58 input[size-1] = 0;
59
60 // Strip person's name
61 input = strchr(input, ':');
62 if (NULL == input)
63 {
64     // Invalid message format
65     return;
66 }
67 input += 2;
68
69 if (0 != strncmp(input, "!calc_", 6))
70 {
71     // Not a calculation command
72     return;
73 }
74
75 _xme_adv_chatCalculator_getResultString
76 (
77     input, output, sizeof(output)
78 );
79
80 xme_core_dcc_sendTopicData
81 (
82     config->publicationHandle,
83     (void*)output, strlen(output)+1
84 );
85 }

```

10. Add the `_xme_adv_chatCalculator_getResultString()` function, which calculates the result of an operation, above `_xme_adv_chatCalculator_receiveDataCallback()`:

```

31 static
32 void
33 _xme_adv_chatCalculator_getResultString(char* in, char* out, int outSize)
34 {
35     // Error messages
36     const char* error_syntax = "The expression could not be evaluated";
37     const char* error_zero = "Division by zero";
38     const char* error_op = "Invalid operator";
39
40     double operand1, operand2, result;
41     char op;
42     char* pos;
43
44     // Parse string
45     pos = strchr(in, 0x20);

```

```

46     if (NULL == pos)
47     {
48         strcpy_s(out, outSize, error_syntax);
49         return;
50     }
51     operand1 = atof(pos+1);
52
53     pos = strchr(pos, 0x20);
54     if (NULL == pos)
55     {
56         strcpy_s(out, outSize, error_syntax);
57         return;
58     }
59     op = *(pos+1);
60
61     pos = strchr(pos, 0x20);
62     if (NULL == pos)
63     {
64         strcpy_s(out, outSize, error_syntax);
65         return;
66     }
67     operand2 = atof(pos+1);
68
69     // Check for operator and division by zero
70     if (!_xme_adv_chatCalculator_isOperationValid(op))
71     {
72         strcpy_s(out, outSize, error_op);
73         return;
74     }
75     if (_xme_adv_chatCalculator_isDivisionByZero(op, operand2))
76     {
77         strcpy_s(out, outSize, error_zero);
78         return;
79     }
80
81     // Calculate result
82     result =
83         _xme_adv_chatCalculator_calculateResult(operand1, op, operand2);
84
85     // Output result
86     sprintf_s
87     (
88         out, outSize, "%lf%c%lf=%lf",
89         operand1, op, operand2, result
90     );
91 }

```

11. Add the following two includes to chatCalculator.c:

```

23  /*****
24  /***   Includes
25  *****/
26  #include "xme/adv/chatCalculator.h"
27
28  #include <float.h>           // Add this line
29  #include <math.h>           // Add this line

```

12. Add the following helper functions above _xme_adv_chatCalculator_getResultString():

```

34  static
35  bool
36  _xme_adv_chatCalculator_isOperationValid(char operation)
37  {

```

```

38     return ('+' == operation || '-' == operation ||
39            '/' == operation || '*' == operation);
40 }
41
42 static
43 bool
44 _xme_adv_chatCalculator_isDivisionByZero(char operation, double operand2)
45 {
46     return ('/' == operation && abs(operand2) < DBL_EPSILON);
47 }
48
49 static
50 double
51 _xme_adv_chatCalculator_calculateResult(double operand1, char operation,
52                                       double operand2)
53 {
54     if ('+' == operation)
55     {
56         return operand1 + operand2;
57     }
58     if ('-' == operation)
59     {
60         return operand1 - operand2;
61     }
62     if ('*' == operation)
63     {
64         return operand1 * operand2;
65     }
66     if ('/' == operation)
67     {
68         // Division by zero is handled in isDivisionByZero()
69         return operand1 / operand2;
70     }
71
72     // Invalid operation is handled in isOperationValid()
73     return 0;
74 }

```

13. Change the text the task sends to the chat room and remove debug output:

```

195 // TODO: Use or remove this sample task callback function:
196 static
197 void
198 _xme_adv_chatCalculator_taskCallback(void* userData)
199 {
200     // TODO: In case you supply the component's configuration instance
201     //       via userData when creating the task, you can access it here:
202     xme_adv_chatCalculator_configStruct_t* config =
203         (xme_adv_chatCalculator_configStruct_t*)userData;
204     /* // TODO: Add code */ // Comment or remove this block
205     // XME_LOG(XME_LOG_NOTE, "Task callback function called!\n");
206
207     // Example: Send data
208     {
209         // Change the following lines
210         static const char* message = "Chat_Calculator_awaiting_commands.\n"
211             "Usage: !calc <operand1> <operation> <operand2>";
212         [...] // Leave the rest of the function as-is
213     }
214 }

```

14. Configure the task to fire every 30 seconds instead of every two seconds and remove debug output:

```

262 xme_core_status_t
263 xme_adv_chatCalculator_activate(xme_adv_chatCalculator_configStruct_t* config)
264 {
265     /* // TODO: Add code */ // Comment or remove this block
266     // XME_LOG(XME_LOG_NOTE, "Activate function of Chat Calculator called!\n");
267
268     // Example: Start a task:
269     config->taskHandle =
270         xme_core_resourceManager_scheduleTask
271         (
272             1000,
273             30000, // Change this line
274             XME_HAL_SCHED_PRIORITY_NORMAL,
275             _xme_adv_chatCalculator_taskCallback,
276             config
277         );
278
279     [...] // Leave the rest of the function as-is
280 }

```

15. Finally, remove some more debug output:

```

288 void
289 xme_adv_chatCalculator_deactivate(xme_adv_chatCalculator_configStruct_t* config)
290 {
291     /* // TODO: Add code */ // Comment or remove this block
292     // XME_LOG(XME_LOG_NOTE, "Deactivate function of Chat Calculator called!\n");
293
294     [...] // Leave the rest of the function as-is
295 }
296
297 void
298 xme_adv_chatCalculator_destroy(xme_adv_chatCalculator_configStruct_t* config)
299 {
300     /* // TODO: Add code */ // Comment or remove this block
301     // XME_LOG(XME_LOG_NOTE, "Destroy function of Chat Calculator called!\n");
302
303     [...] // Leave the rest of the function as-is
304 }

```

16. That's it! Compile and run the **tutorial** example (press *F7* and *F5* in *Visual Studio*). Test the new component by letting the *Chat Calculator* do your math homework ;)

7 More Examples

You might consider implementing the following components to get to know CHROMOSOME even better:

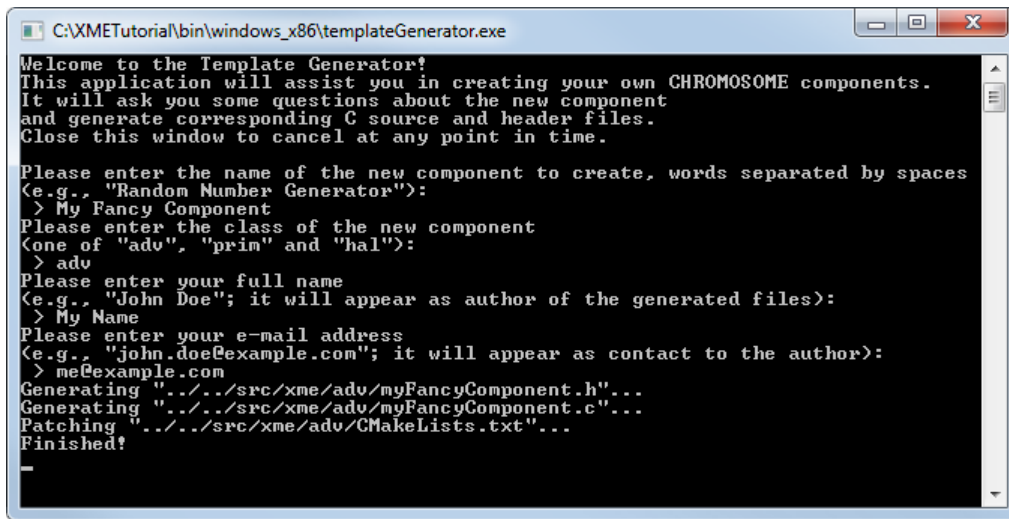
- A “spammer” component that sends unsolicited advertisements to chat channels.
- A component that keeps track of all persons in each chat room and supports a “!who” command to print a list (you might want to extend the *Chat Example* to publish additional data in order to do this).
- A “lunch time” component that announces lunch time at noon in every chat room.

A Frequently Asked Questions

A.1 How can I create a new component?

CHROMOSOME comes with an application that can create template files for new components. This application is called `templateGenerator` and can be found in the `bin/windows_x86` sub-directory of the release package or built manually just like any other CHROMOSOME example program (see Section 4 for hints).

The application will ask you for the component name, type as well as your name (for the author field) and generate a header and a source file for the new component in the respective directory (compare Figure 16). Consider using one of the existing example applications for testing your new component if application, see Section 6 for more information on how to embed a new component into an existing application.



```
C:\XMETutorial\bin\windows_x86\templateGenerator.exe
Welcome to the Template Generator!
This application will assist you in creating your own CHROMOSOME components.
It will ask you some questions about the new component
and generate corresponding C source and header files.
Close this window to cancel at any point in time.

Please enter the name of the new component to create, words separated by spaces
(e.g., "Random Number Generator"):
> My Fancy Component
Please enter the class of the new component
(one of "adv", "prim" and "hal"):
> adv
Please enter your full name
(e.g., "John Doe"; it will appear as author of the generated files):
> My Name
Please enter your e-mail address
(e.g., "john.doe@example.com"; it will appear as contact to the author):
> me@example.com
Generating "../src/xme/adv/myFancyComponent.h"...
Generating "../src/xme/adv/myFancyComponent.c"...
Patching "../src/xme/adv/CMakeLists.txt"...
Finished!
```

Figure 16: *Template Generator* in action.

A.2 How can I define a new topic?

Topic identifiers are simple numbers with associated semantics. The `xme_core_topic_t` type defined in file `<XME_ROOT>/xme/core/topic.h` declares some generic topic identifiers. Topic numbers in the range between zero and `XME_CORE_TOPIC_USER-1` are reserved for internal use. This means that you can define a new topic by using an arbitrary number larger or equal to `XME_CORE_TOPIC_USER`.

Note that there is currently no mechanism in place for checking that the same topic identifiers are always used for the same type of data. Be aware that other people in the same network might use the same topic identifiers for their data. We plan to put a mechanism in place for avoiding this issue in future versions of CHROMOSOME.

A.3 How can I port CHROMOSOME to my own target platform?

The current release of CHROMOSOME supports only Windows. However, we are in the process of developing platform support layers for other platforms. These include Linux, medium-sized microcontroller platforms like ARM Cortex M3 (e.g., as central control units) as well as low-end microcontrollers like Atmel AVR (e.g., for sensor networks). Upcoming releases of CHROMOSOME will ship with the respective functionality.

If you favorite platform is not on this list, there are multiple options:

1. You can suggest **CHROMOSOME** to be ported onto your platform. If your application is of high relevance, we might indeed port **CHROMOSOME** to that platform.
2. You can contribute to the development of **CHROMOSOME** and have your platform supported as a target platform.
3. You can take the **CHROMOSOME** source code and implement the respective platform support without revealing the source code. Our license model permits this (see also [Appendix D](#)).

B Installing Visual C++ 2010 Express

Follow these steps to install Visual C++ 2010 Express:

1. Point your favorite browser to <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express> (compare Figure 17).

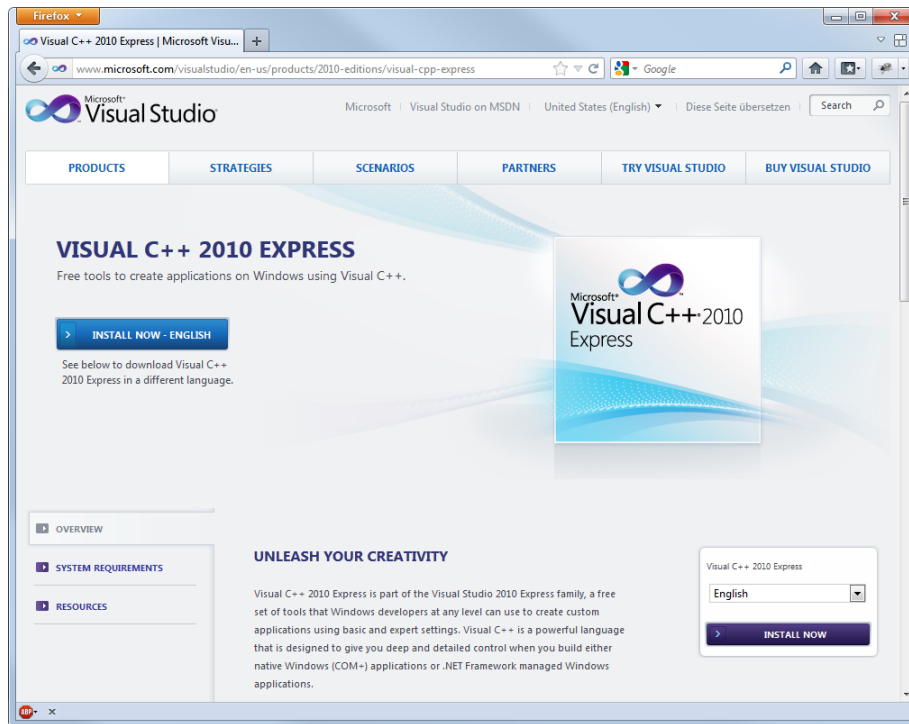


Figure 17: Selecting Visual C++ 2010 Express for download.

2. Choose your language and click on *Install*.
3. If you are asked whether you want to install Visual Studio 2011 Professional instead, choose *Visual C++ 2010 Express (English)* (compare Figure 18)¹³.
4. After downloading `vc_web.exe`, launch it. Read the welcome screen and click *Next* when ready.
5. Read the terms and conditions and choose the appropriate option.
6. On the *Installation Options* page, you may deselect *Silverlight* and *SQL Server 2008*, they are not needed for CHROMOSOME (Figure 19).
7. On the *Destination Folder* page, select the installation directory. In some cases, you might not be able to choose a directory manually.
8. Wait for *Visual C++* setup to finish downloading and installation.
9. After a few minutes, *Visual C++* setup will report that the installation has finished. In some cases, a reboot may be required to use *Visual C++*.

¹³ Alternatively, you can use the following direct link:
http://download.microsoft.com/download/1/D/9/1D9A6C0E-FC89-43EE-9658-B9F0E3A76983/vc_web.exe

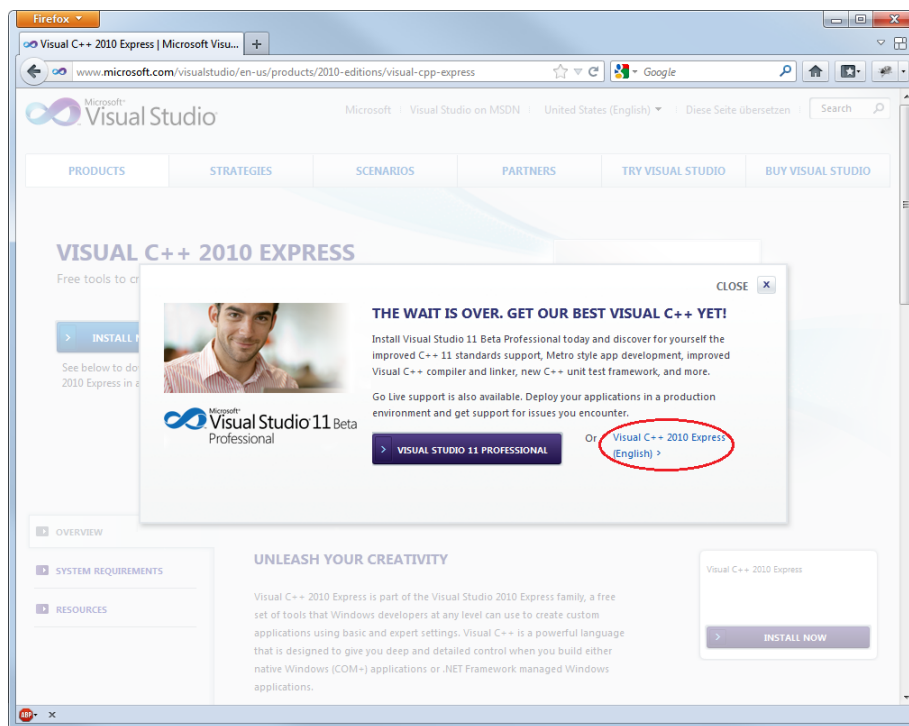


Figure 18: Downloading Visual C++ 2010 Express, highlighted in red the download link.

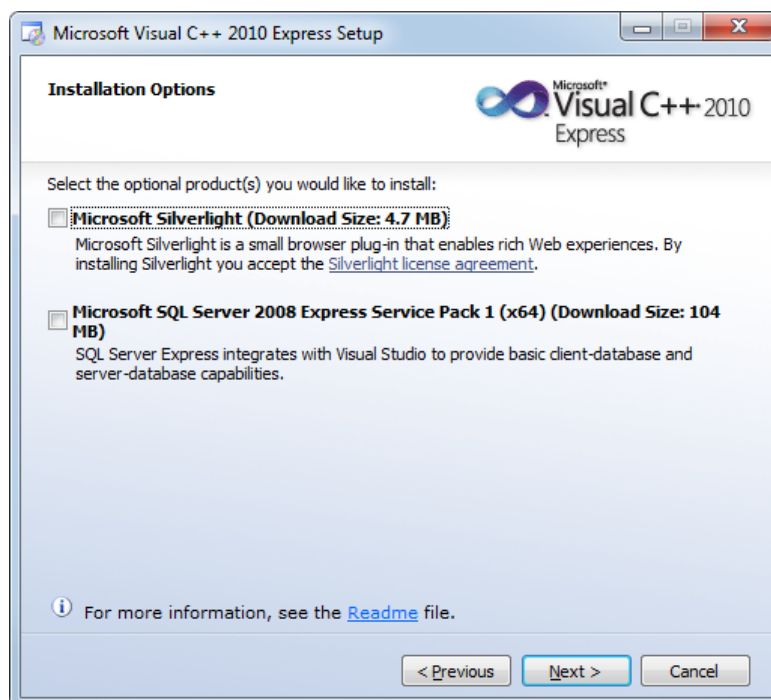


Figure 19: Visual C++ 2010 Express installation options.

C Installing CMake

Follow these steps to install CMake:

1. Point your favorite browser to <http://cmake.org/cmake/resources/software.html> and click on the link corresponding to the Windows installer (compare Figure 20).

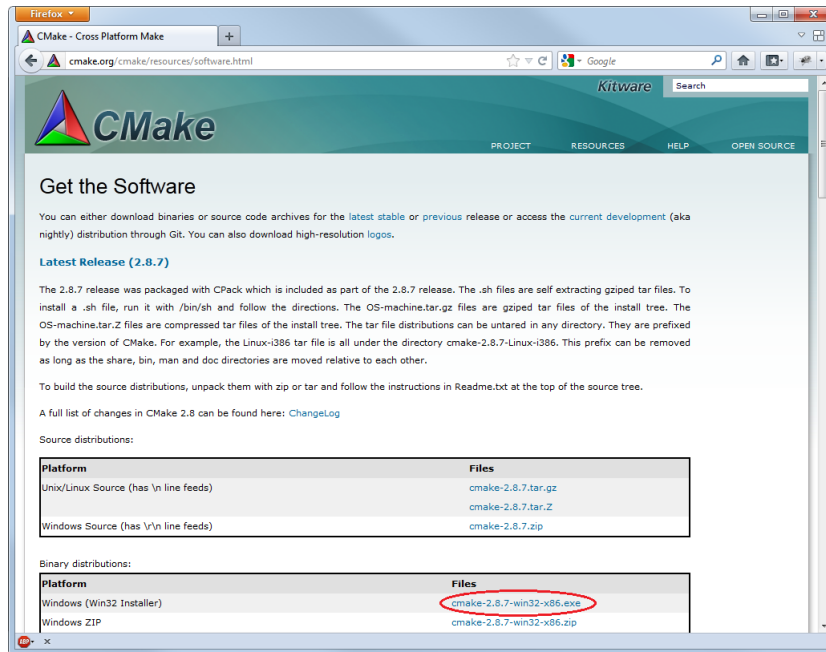


Figure 20: Selecting CMake for download, highlighted in red the download link.

2. After downloading the setup, launch it. Follow the instructions on the screen. When you get prompted whether to add CMake to the system PATH, you may choose to *not* add it (compare Figure 21). CHROMOSOME does not require CMake to be on the system search path.

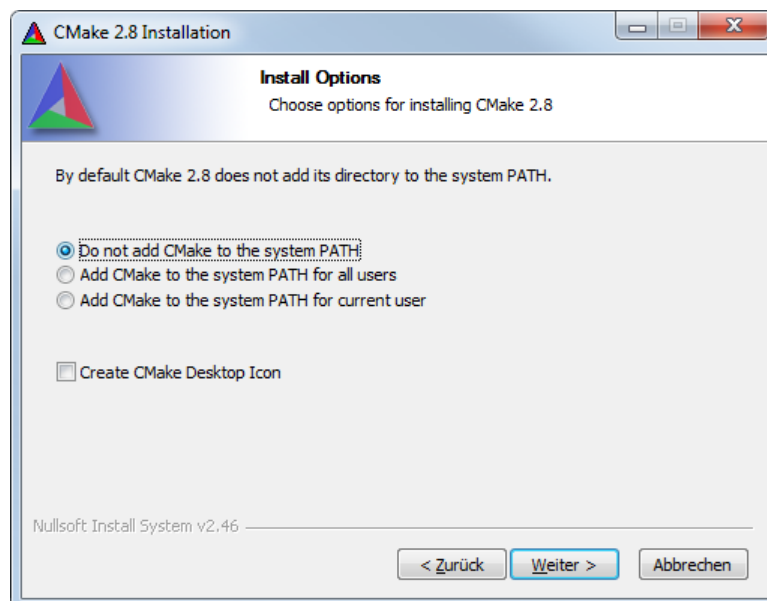


Figure 21: CMake system PATH options.

D CHROMOSOME License

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and

may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier

identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.