

Pitch pRTI™ USER'S GUIDE

v 5.3

Contents

1	INTRODUCTION	5
1.1	ABOUT THIS DOCUMENT	5
1.2	ABOUT PITCH TECHNOLOGIES	6
1.3	ABOUT IEEE 1516 HLA	6
1.4	ABOUT THE RUN TIME INFRASTRUCTURE	7
1.5	ABOUT PITCH PRTI™	7
1.6	PRODUCT LICENSING STRUCTURE	8
2	PREPARING FOR PITCH PRTI™	9
2.1	A TOPOLOGY AND COMPONENTS EXAMPLE	9
2.2	THE CENTRAL RTI COMPONENT (CRC)	10
2.3	THE FEDERATES	10
2.4	THE LOCAL RTI COMPONENT (LRC)	10
2.5	FOM FILES	10
2.6	THE COMPUTERS	11
2.7	NETWORKING	12
2.8	POWERFUL USER INTERFACES	12
3	INSTALLING PITCH PRTI™	16
3.1	WINDOWS INSTALLATION	16
3.2	ACTIVATING LICENSES	22
3.3	VERIFYING THE WINDOWS INSTALLATION	24
3.4	LINUX INSTALLATION	26
3.5	VERIFYING THE LINUX INSTALLATION	26
3.6	INSTALLING ON OTHER PLATFORMS	28
4	UNINSTALLING PITCH PRTI™	29
4.1	UNINSTALLING ON WINDOWS	29
4.2	UNINSTALLING ON LINUX	29
4.3	UNINSTALLING ON OTHER PLATFORMS	29
5	RUNNING PITCH PRTI™	30
5.1	GRAPHICAL USER INTERFACES OVERVIEW	30
5.2	USING THE GRAPHICAL USER INTERFACE	30
5.3	USING THE COMMAND LINE INTERFACE	42
6	RUNNING PITCH PRTI™ IN SERVICE-MODE	43
6.1	INTRODUCTION	43
6.2	LIMITATIONS	43
6.3	INSTALLING THE SERVICE	43
7	USING PITCH WEB VIEW	45
7.1	INTRODUCTION	45
8	PITCH CONTROL CENTER	47
8.1	OVERVIEW	47
8.2	FRIENDLY ERRORS	47
8.3	LRC MONITORING	48
9	DEVELOPING WITH PITCH PRTI™	53
9.1	MICROSOFT VISUAL STUDIO 2010 ON WINDOWS	53
9.2	OPTIMIZATION FLAGS FOR VISUAL STUDIO RELEASE BUILDS	58
9.3	RUNNING FEDERATES FROM THE VISUAL STUDIO IDE	58
9.4	OTHER MICROSOFT VISUAL STUDIO VERSIONS ON WINDOWS	59

9.5	GCC ON LINUX	59
9.6	CUSTOM SIGNAL HANDLERS IN C++	60
9.7	JAVA	60
10	WRITING A SIMPLE FEDERATE IN C++	61
10.1	THE FEDERATEAMBASSADOR AND THE RTIAMBASSADOR	61
10.2	CONNECTING TO THE RTI	62
10.3	THE FEDERATION OBJECT MODEL	62
10.4	PUBLISHING AND SUBSCRIBING TO INFORMATION	63
10.5	SENDING INTERACTIONS	63
10.6	RECEIVING INTERACTIONS	64
10.7	CONCLUSIONS	64
11	WRITING A SIMPLE FEDERATE IN JAVA	65
11.1	THE FEDERATEAMBASSADOR AND THE RTIAMBASSADOR	65
11.2	CONNECTING TO THE RTI	66
11.3	THE FEDERATION OBJECT MODEL	66
11.4	PUBLISHING AND SUBSCRIBING TO INFORMATION	67
11.5	SENDING INTERACTIONS	67
11.6	RECEIVING INTERACTIONS	68
11.7	CONCLUSIONS	68
12	TICK AND PROCESS MODELS	70
12.1	SETTING THE PROCESS MODEL	70
12.2	PRACTICAL GUIDELINES	70
12.3	EXPLANATION OF PROCESS MODELS	70
13	DEBUGGING AND TRACING	72
13.1	OVERVIEW	72
13.2	ENABLING THE TRACING	73
13.3	FORMAT OF THE TRACE LOG	73
13.4	A SAMPLE TRACE LOG	73
14	NETWORKING	75
14.1	WHEN TO RECONFIGURE NETWORKING	75
14.2	OVERVIEW OF PITCH PRTI™ COMMUNICATION	76
14.3	USING MULTICAST	77
14.4	OPERATING OVER FIREWALLS	77
14.5	NETWORK SETTINGS	78
14.6	PITCH PRTI™ AND PITCH BOOSTER™	79
15	ADVANCED PITCH PRTI™ NETWORK PERFORMANCE TUNING	82
15.1	INTRODUCTION	82
15.2	FEDERATE TUNING	82
15.3	SETTING TUNING PARAMETERS	83
15.4	PITCH PRTI™ TUNING ALGORITHMS	87
16	CONFIGURATION REFERENCE	91
16.1	SETTINGS FOR THE CENTRAL RTI COMPONENT	91
16.2	SETTINGS FOR THE LOCAL RTI COMPONENT	94
16.3	OLDER APIS SETTINGS	105
16.4	OVERRIDES SETTINGS	107
16.5	LRC JVM SETTINGS FOR C++ FEDERATES	108
17	USING FEDERATES DEVELOPED FOR LEGACY HLA VERSIONS	109
17.1	C++ LIBRARIES	109
17.2	HEADERS	109

17.3	JAVA LIBRARIES	110
17.4	TIME CLASSES	110
17.5	DATA DISTRIBUTION MANAGEMENT (DDM) SERVICES	112
17.6	QUICK REFERENCE	113
18	COMMON ERRORS	114
18.1	COMPILING C++ FEDERATES	114
18.2	COMPILING JAVA FEDERATES	115
18.3	STARTING PITCH PRTI™	115
18.4	RUNNING C++ FEDERATES	116
18.5	FEDERATION STARTUP	117
18.6	GET HANDLES AND REGISTER OBJECT INSTANCES	118
18.7	UPDATES AND INTERACTIONS	120
18.8	TIME MANAGEMENT	121
18.9	MISCELLANEOUS	121

1 Introduction

1.1 About This Document

Pitch pRTI™ and Pitch Visual OMT™ are registered trademarks belonging to Pitch Technologies AB. All rights reserved.

This document provides information about how to install, run, troubleshoot and optimize Pitch pRTI™, the leading commercial RTI for the HLA. The acronym RTI stands for *Run-Time Infrastructure*, pRTI™ stands for *portable RTI* and HLA stands for *High Level Architecture*, which is a standard for simulation interoperability.

These are the main audiences for this document:

- **Developers** of HLA compliant simulation system who wish to use Pitch pRTI™ in their development projects.
- **IT Staff** responsible for setting up and maintaining HLA based simulation applications in their IT environment. This group can concentrate on chapters 2, 3 and 5.
- **Technical specialist** who wish to evaluate Pitch pRTI™ as an interoperability infrastructure for their projects or products.

The outline of the document is as follows:

- This **introduction** gives an initial overview of HLA and Pitch pRTI™.
- **Preparing for Pitch pRTI™** describes the set-up you need to run and develop for pRTI™, such as networking, hardware, software and participating systems.
- **Installing Pitch pRTI™** describes how to install pRTI™ on various platforms such as Windows, Linux etc.
- **Running Pitch pRTI™** describes how to run and monitor your simulations using a graphical and command line interface.
- **Running Pitch pRTI™ In Service-Mode** describes how to install and setup the pRTI™ Central RTI Component to run as a service on Windows and Linux.
- **Using Pitch Web** describes how to access the Pitch pRTI™ using web browsers.
- **Developing with Pitch pRTI™** shows how to set up your development environment to be able to build federates.
- **Writing a simple federate in C++** contains an example federate complete with C++ source code including instructions on how to compile and run the federate.
- **Writing a Simple Federate in Java** contains an example federate complete with Java source code including instructions on how to compile and run the federate.
- **Tick and Process Model** describes how the RTI and the federate share the CPU and how to achieve optimal performance and responsiveness.

- **Debugging and Tracing** describes the features of pRTI™ to assist you in debugging your federation.
- **Networking** describes networking functionality and tuning for pRTI™.
- **Advanced Pitch pRTI™ Network Performance Tuning** describes the advanced network tuning functionality available in pRTI™.
- **Configuration Reference** contains a summary of all the configuration switches that can be used with pRTI™.
- **Common errors** lists common errors and how to resolve them.

1.2 About Pitch Technologies

Pitch Technologies is the world leading supplier of interoperability enabling products for simulation and training. Based on open international standards, Pitch provides COTS products for developing and deploying distributed simulations according to the High Level Architecture (HLA). Our products Pitch pRTI™, Pitch Visual OMT™, Pitch Commander™, Pitch Recorder™, Pitch Developer Studio™, Pitch Booster™, Pitch Extender™ and Pitch DIS Adapter™ are used in a variety of both civilian and defense simulations world-wide to support training, acquisition and analysis. Furthermore, many of the simulation industry's vendors use the Pitch products to HLA enable their solutions.

1.3 About IEEE 1516 HLA

The HLA standard was initially defined in 1995 based on experiences from earlier simulation interoperability standards. The idea was to create a standard that could embrace many domains and types of simulation. Although earlier standards existed they were limited to specific simulation domains or did not provide services for managing time in simulations.

The HLA standard was developed in several steps from 1.0 up to the HLA 1.3 standard. After this step it was decided to broaden the usage outside the US defense, so the current version of the standard is established as an open and international IEEE standard: the IEEE 1516 HLA standard. The IEEE 1516 HLA standard is the intended goal also for the US Department of Defense simulations.

IEEE accepted the standard in 2000. Since then the development of tools and RTI:s for this standard has started. The first complete RTI implementation, Pitch pRTI™ 1516, was released in December 2001. Other HLA tools available include object model tools such as Pitch Visual OMT™ and HLA data loggers such as Pitch Recorder™. Today there is a wide range of tools from several vendors available on the market.

Starting with version 2.1, Pitch pRTI™ 1516 was based on the IEEE standards and the DMSO interpretations version 2.

Starting with version 4.2, Pitch pRTI™ was based on the latest improved version of the HLA standard, IEEE 1516-2010, also known as *HLA Evolved*. You may read more about HLA Evolved in the "Papers" section on our web site www.pitch.se.

The IEEE 1516 standard is in active use in Europe, Asia and the US. Worth noting is the acceptance by the non-defense community. Commercial applications based on the IEEE 1516 standard have already been completed and delivered to the end user.

HLA lets you interconnect simulations, devices and humans in a common federation. HLA builds on composability, letting you construct simulations from pre-built components.

Each computer based simulation system is called a *federate* and the group of interoperating systems is called a *federation*.

The HLA standard consists of three parts:

- The HLA rules that the entire federation and federates have to follow.
- The Object Model Template, which is used to describe object models for federates and federations. The leading graphical tool for working with such object models is Pitch Visual OMT™ from Pitch Technologies.
- The HLA Interface Specification, which describes the functionality that the RTI has to provide. .

To be able to successfully develop HLA compliant applications it is necessary to gain a deeper understanding of HLA than this document provides. We recommend reading the HLA Tutorial, available for free from www.pitch.se or Hands-On HLA training available from Pitch. You may also want to study the FEDEP/DSEEP, which is the recommended process for developing federations.

1.4 About the Run Time Infrastructure

The *Run Time Infrastructure* is responsible for the information exchange during the execution. It will let federates join and resign, declare their intent to publish information, send information about objects, attributes and interactions, synchronize time, etc. Note that the HLA is not the RTI but it specifies that there must be an RTI with a standardized interface. The interface is specified in the HLA standard, but the implementation of the interface specification is left to the RTI developer.

1.5 About Pitch pRTI™

The product Pitch pRTI™ is an implementation of the IEEE 1516 Interface Specification. It lets you integrate simulations in an HLA compliant way. You can mix different operating systems and programming languages. The main advantages of Pitch pRTI™ are:

- **It is complete, certified and HLA compliant.** Pitch pRTI™ is a complete implementation of the HLA specification.
- **It offers excellent performance.** Pitch pRTI™ features sender-side filtering for updates and interactions, which will substantially reduce network and CPU load in large federations. You may also reduce the workload of your federates using advisories which impose very little overhead. Still Pitch pRTI™ has very modest CPU and memory requirements.
- **It provides advanced debugging capabilities.** There is an extensive GUI that allows you to inspect the state of your federation during runtime as well as a powerful set of debugging tools.
- **It is easy to install and run.** Pitch pRTI™ is extremely easy to install and configure. It is easy to mix various platforms and languages in the same federation.

- **It runs well over the Internet and other Wide Area Networks.** By adding Pitch Booster™ at each site, it is easy to run HLA simulations using Pitch pRTI™ across both LAN and the Internet. Read more in section 14.6.
- **It is network-friendly.** You can do optimizations and configurations for LAN, WAN and firewalls and inspect the status of the network graphically.
- **It is commercially packaged and supported.** It is possible to get an OEM license to include it with your products. You can get local support and consulting in several countries from Pitch or one of our distributors. See <http://www.pitch.se> for details.
- **It is superior for long-running federations.** It handles unreliable federates gracefully including automatic resign, ownership and time management recovery and more.
- **It gives you the ability to integrate your existing C/C++ simulators with platform-independent Java systems.** Pitch pRTI™ provides API:s for both C++ and Java, so you can use federates written in any of those languages together in the same federation.

Pitch pRTI™ is developed by Pitch. Professional consulting services and training is also available.

1.6 Product Licensing Structure

The product is structured in the following way:

The Pitch pRTI™ base Central RTI Component (CRC), a.k.a. RTIexec license enables you to run Pitch pRTI™ with a certain number of federates. You may for example purchase a license for 10 federates. This will enable you to connect up to 10 federates (simulation systems) together. The federates may run on the same computer or several different computers. Note that the Web Services API for HLA Evolved is licensed separately.

Starting with v 4.5 it is also possible to combine multiple licenses whose number of allowed federates are added to the total sum of allowed federates.

In addition to this, there is also local federate licensing which means that federates can bring own licenses to the CRC. Such federates will not consume any license from the CRC license.

2 Preparing for Pitch pRTI™

This chapter describes the computer hardware and software that is needed to run a simulation using Pitch pRTI™. It also describes various configurations that are possible.

2.1 A Topology and Components Example

An example of a federation of computer-based simulations that interoperates using Pitch pRTI™ is described in the Figure 1

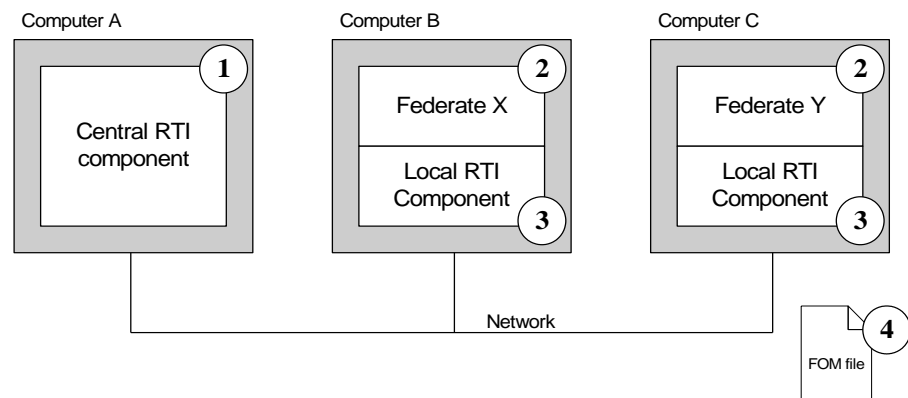


Figure 1 – An example topology.

The environment in Figure 1 consists of:

1. The *Central RTI Component* that manages the federation.
2. The *federates* participating in the federation.
3. The *Local RTI Component* which each federate use to communicate in the federation.
4. One or more *FOM modules* that describes the Federation Object Model.

The federates and the Central RTI Component are running on separate computers. Note that this is not required. Any number of federates can run on the same computer, and any number of federates can run on the same computer as the Central RTI Component. Another possible topology that shown in Figure 2.

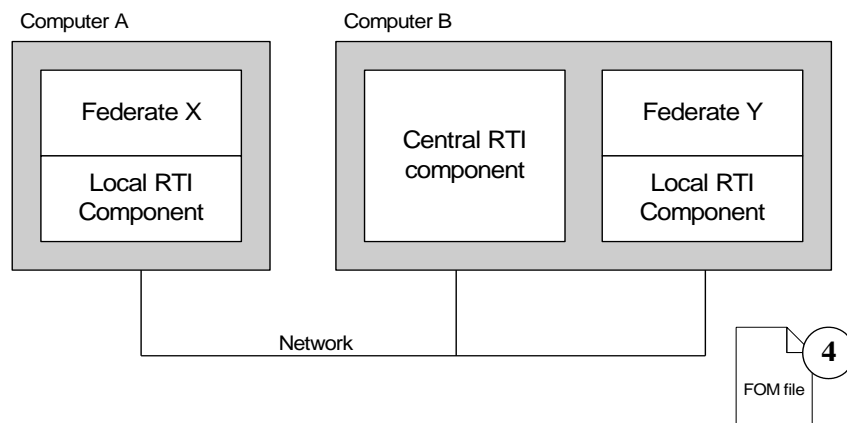


Figure 2 - Another example topology.

2.2 The Central RTI Component (CRC)

The CRC is the central component of Pitch pRTI™, also known as the RTIexec. It provides a graphical and a command line interface that lets you monitor the execution. It is responsible for coordinating the entire federation and distributes the work between the Local RTI Components.

When a federate wants to join a federation execution, it connects to the CRC and receives information about the federation execution such as which other federates are currently joined to the federation and how to communicate with them.

2.3 The Federates

The federates may be implemented in several programming languages:

- C++
- Any programming language with an interface module written in C++
- Java
- Any programming language, wrapped in Java

You may mix different implementation languages in the same federation.

2.4 The Local RTI Component (LRC)

Every federate is compiled and linked with a component that contains the classes and methods that are used to connect to the federation. This component is called the Local RTI Component (LRC). The LRC takes care of the federate's need to exchange information with other federates.

This makes Pitch pRTI™ a distributed application, consisting of both the CRC and a number of LRC:s (one for each federate). Figure 3 illustrates the relationship between the CRC, the LRC and the federates.

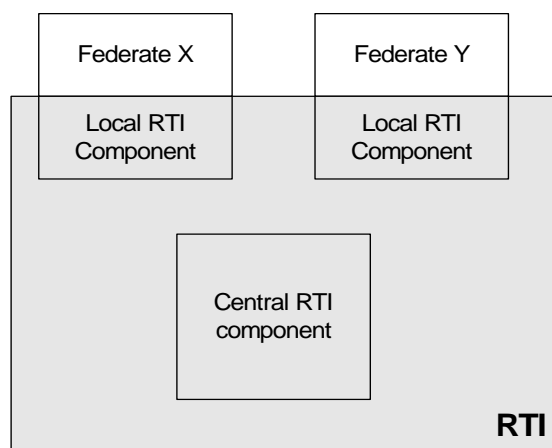


Figure 3 – The relationship between the CRC, the LRC and the federates.

2.5 FOM Files

The FOM files contain the Federation Object Model. The Federation Object Model describes the information exchange in the federation. This includes objects, interactions, attributes, parameters and data types for all the information that is

exchanged between the federates. The data in the files is XML formatted, as specified by the HLA standard. A new feature of the HLA Evolved standard is the concept of modular FOMs. This means that the federation object models can be composed by multiple FOM modules, which can contain either the entire FOM, extensions to another FOM module, or new concepts independent of other FOM modules.

Federation object models can be edited using Pitch Visual OMT™. You can use it to create, edit and manage Federation Object Models. Figure 4 shows a screenshot of the tool.

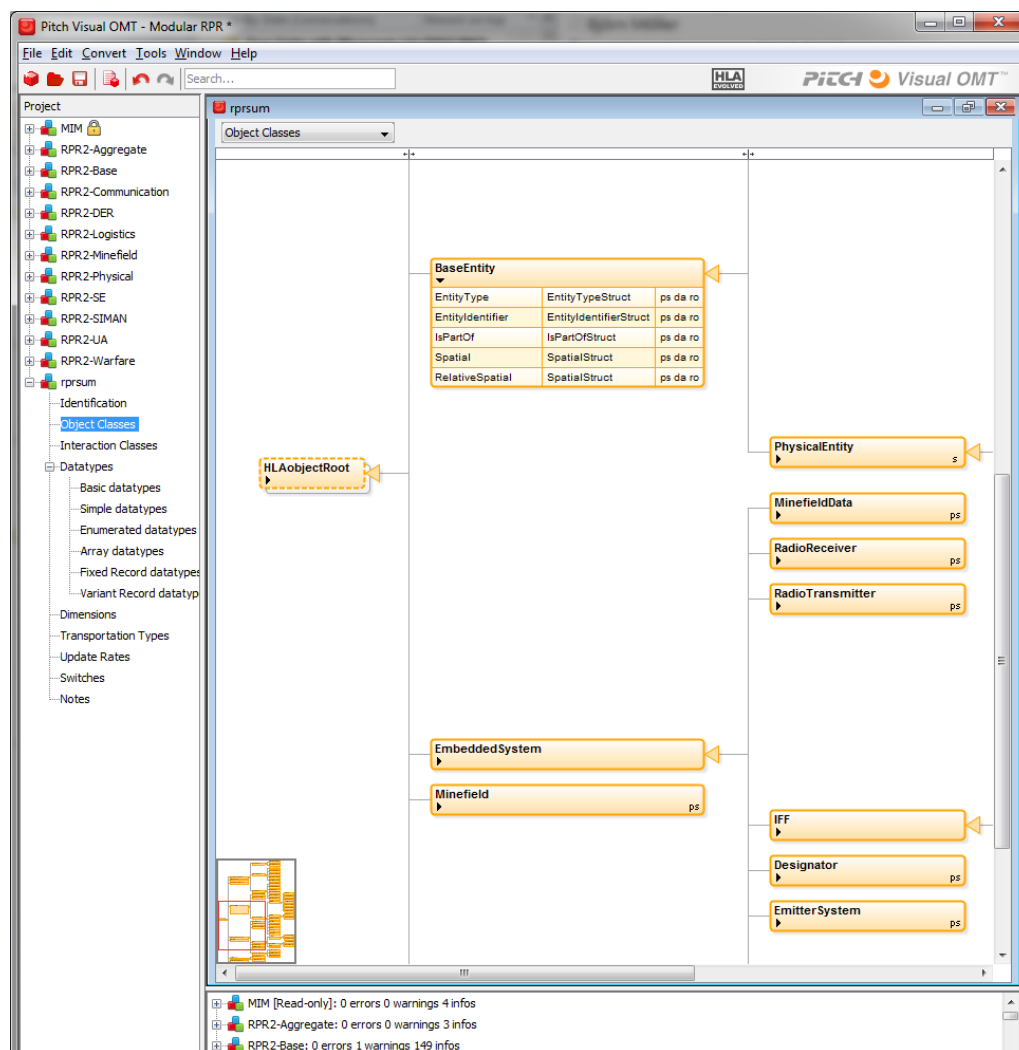


Figure 4 – Pitch Visual OMT™ object modeling tool.

2.6 The Computers

The computers that you use will have to provide appropriate networking according to the next section as well as meet the requirements of the local federates. Supported operating systems include:

- Windows XP/2003/Vista/7/2008 using both 32 and 64 bit architectures
- Linux on x86 and x64 architectures
- Mac OS X 10.6, 10.7, 10.8

- Other platforms may be supported on demand

You may mix different brands and operating system in the same federation. Additional platforms may be available and customer ports can be performed. Contact Pitch or your local distributor for more information.

2.7 Networking

The protocol used for Pitch pRTI™ is the industry standard TCP/IP. The most commonly used network configurations for simulations are:

- An Ethernet-based Local Area Network (LAN). All computers are equipped with Ethernet cards and connected to a hub or switch with 10/100/1000 Base-T cables. Note that using a switch may significantly improve performance for federations exchanging large amounts of data.
- WiFi. All computers are equipped with a WiFi card and establish wireless connections through a WiFi router or access point.
- All federates run on a single computer, which is not connected to a network. In this case you will a loop-back network interface.

In addition to this you will need to be able to specify the address of the CRC or RTIexec to which you are trying to connect using one of the following:

- If you have a DNS service it will provide the translation between names (such as myhost.pitch.se) and IP addresses (such as 192.168.1.1). In this case you will need to know the name of the computer running Pitch pRTI™. Note that the DNS server should also be able to resolve IP addresses to names, also known as reverse lookups.
- If you do not have a DNS service you can use the IP address directly.
- If you are using a loop-back interface, the standard address is 127.0.0.1.
- If you are using Pitch pRTI™ over Pitch Booster™ a unique CRC-name is used to address the CRC.

Contact your network administrator for details about your organization's network.

The default Pitch pRTI™ settings are suitable for both local area networks and wide area networks. Pitch pRTI™ runs well over WAN:s with routers, for example over the Internet or big corporate networks. See chapter 14 for more information.

2.8 Powerful User Interfaces

Pitch pRTI provides three graphical user interfaces (GUIs):

- The Desktop GUI that facilitates the overall management of the federation and federates.
- The Pitch Control Center, which facilitates monitoring and troubleshooting locally on a computer running individual federates.
- The Web View user interface that can be accessed from any computer, tablet or mobile phone on the network using a web browser.

These three GUIs are provided for different purposes, as described below.

The Desktop GUI is available when running the CRC on a desktop. It is shown in Figure 5.

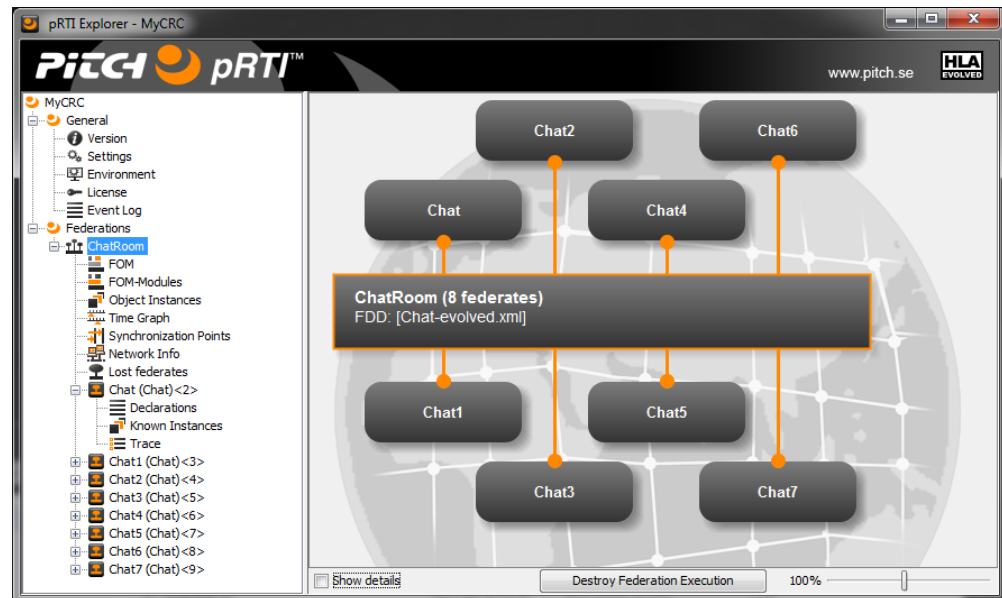


Figure 5 –Desktop CRC GUI

This GUI is targeted at users that are responsible for managing the entire federation. It enables users to monitor which federations and federates that are currently available, their overall state. It also provides information about the Federation Object Models used, registered objects, publication, subscriptions, ownership, time management and more.

Pitch Control Center is available on any computer running a federate. It is shown in the following figure.

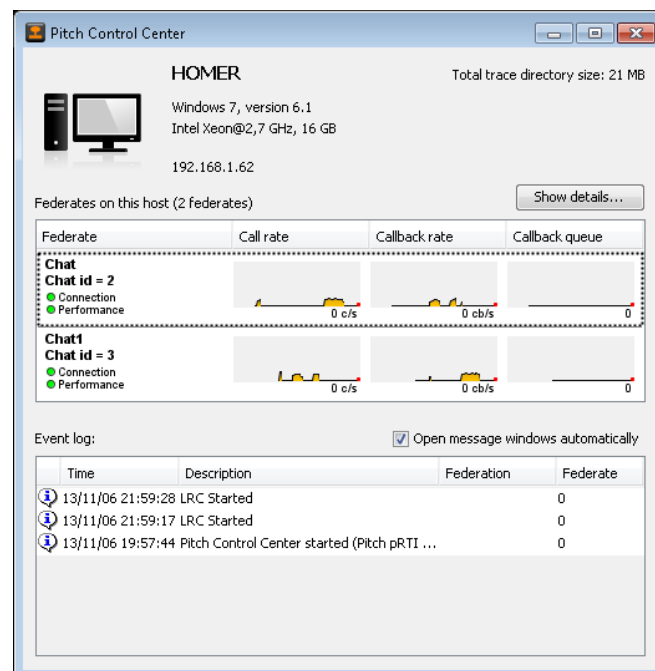


Figure 6 –Pitch Control Center

This GUI is targeted at developers and users that need to monitor and troubleshoot individual federates. It provides monitoring of the connection status and performance, with graphical monitoring of incoming and outgoing call rates and queues. It also provides Friendly Errors, whereby error messages can be delivered to a user of federate problems, without interfering with the federate itself. Friendly Errors also includes suggestions for solving configuration problems.

The **Web View** is available on any computer, tablet or mobile phone on the network with a web browser. It is shown in the following figure.

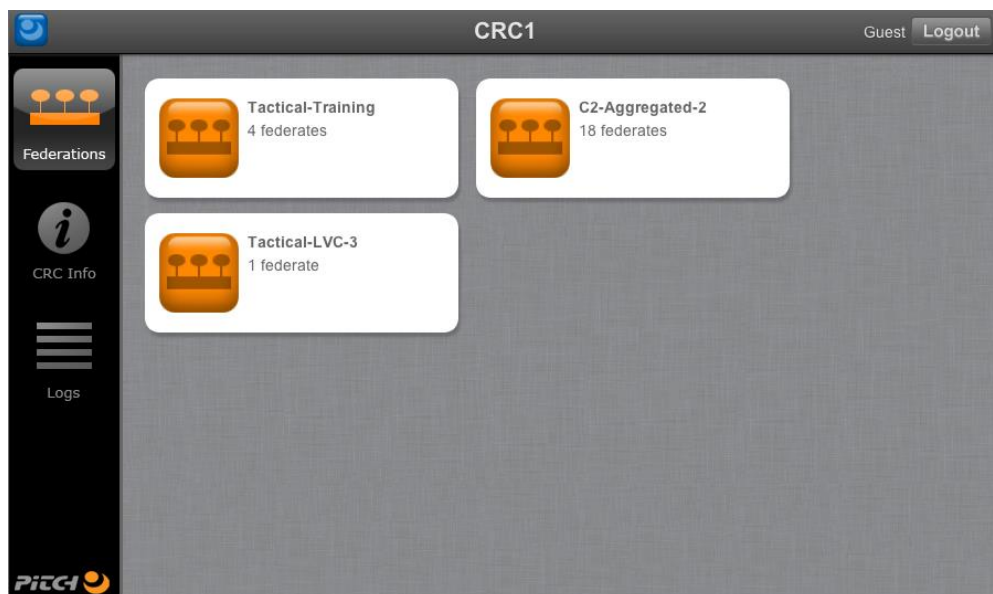


Figure 7 –Pitch Web View

This GUI enables federation managers, federate developers and IT staff to quickly get an insight into the RTI from anywhere and from any device. Several users can connect at the same time.

Different user levels can be used. “Federation Manager” users may resign federates and may destroy federation executions but a “Guest” user may only be able to see the current status of the federation and the joined federates.

The Web View is implemented as a web application that connects to the Central RTI Component.

3 Installing Pitch pRTI™

This chapter covers how to install and verify the Central RTI Component of Pitch pRTI™ using the sample federates. It also covers the installation of the Local RTI Components for the use by your own federates.

3.1 Windows Installation

Before you start, check that you have:

- The installer executable.
- Your license activation key that you will use to activate the software, in case of installing a CRC or a locally licensed LRC.

Launch the installer executable. A graphical installer will now start.

Figure 8 shows the introduction screen which gives you a general introduction to the installation.

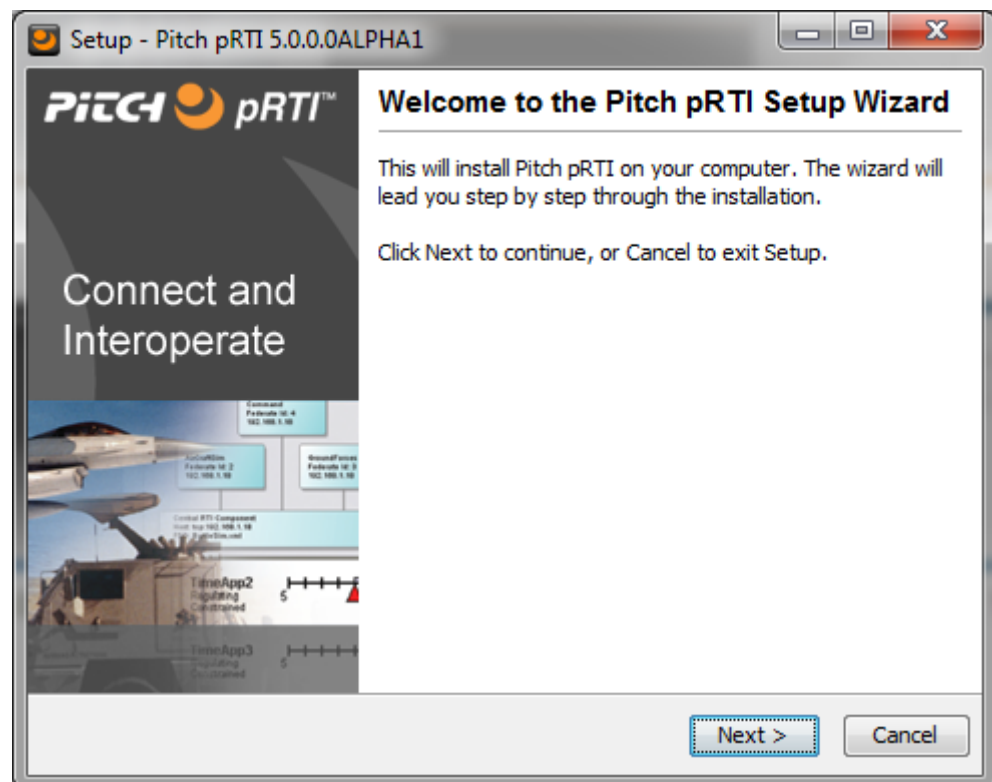


Figure 8 – Installation introduction.

Click *Next* to continue. The license agreement will then be presented to you as shown in Figure 9.

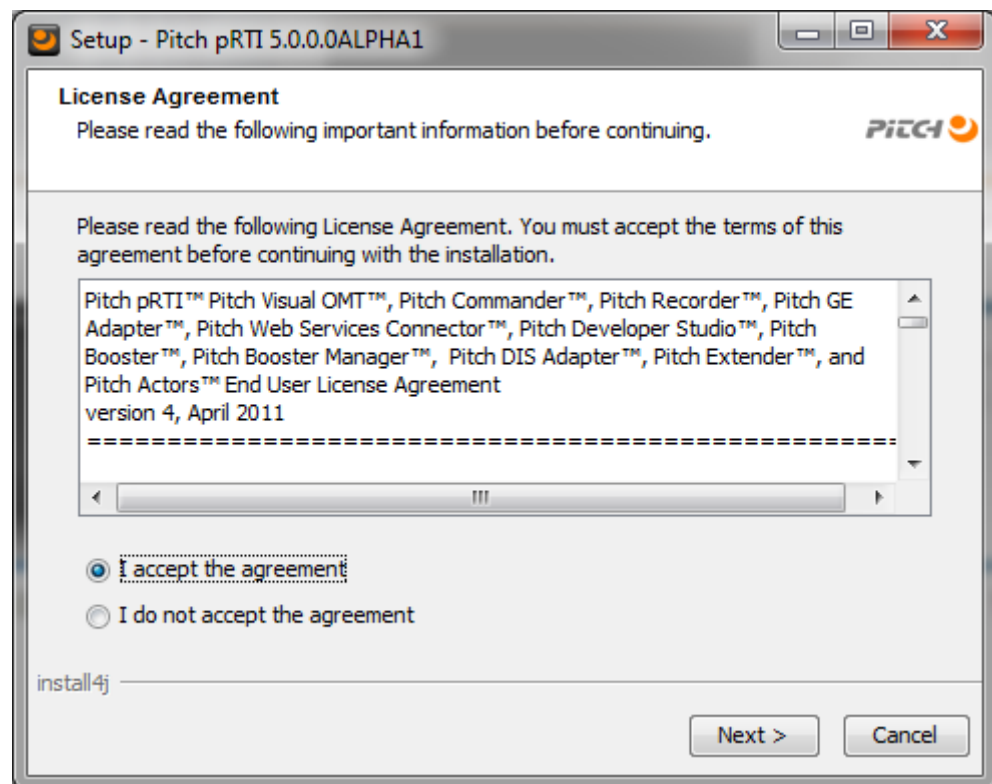


Figure 9 – End user license agreement.

Read the license agreement and make the appropriate selection. Then click *Next*.

You are now supposed to select where to install pRTI™ on your system. You are recommended to use the default installation directory, but pRTI™ will work properly even if it is installed in a different directory.

Choose installation directory for Pitch pRTI™

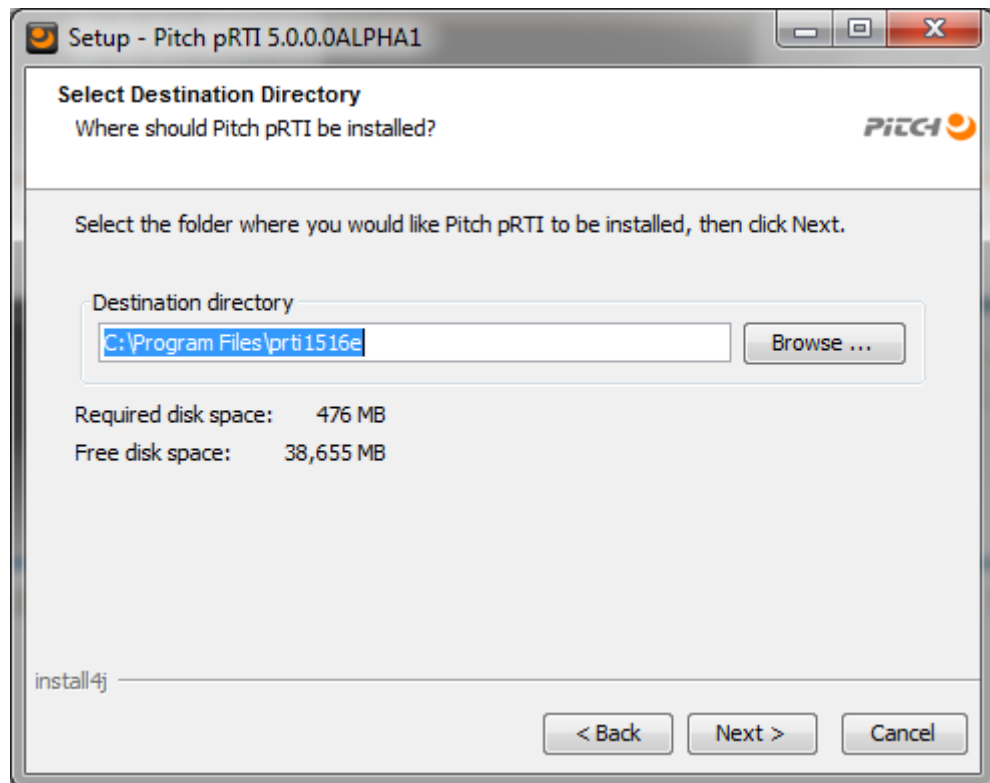


Figure 10 – Installation destination.

You can now select which type of installation you want. The default selection installs all components, including both the CRC and the LRC. If you are installing on a computer that will run the RTIexec (the CRC) we recommend both options. If you are installing on a computer where only federates will be running, you only need the LRC option.

The *Web View Server* component is also optional and could be installed standalone.

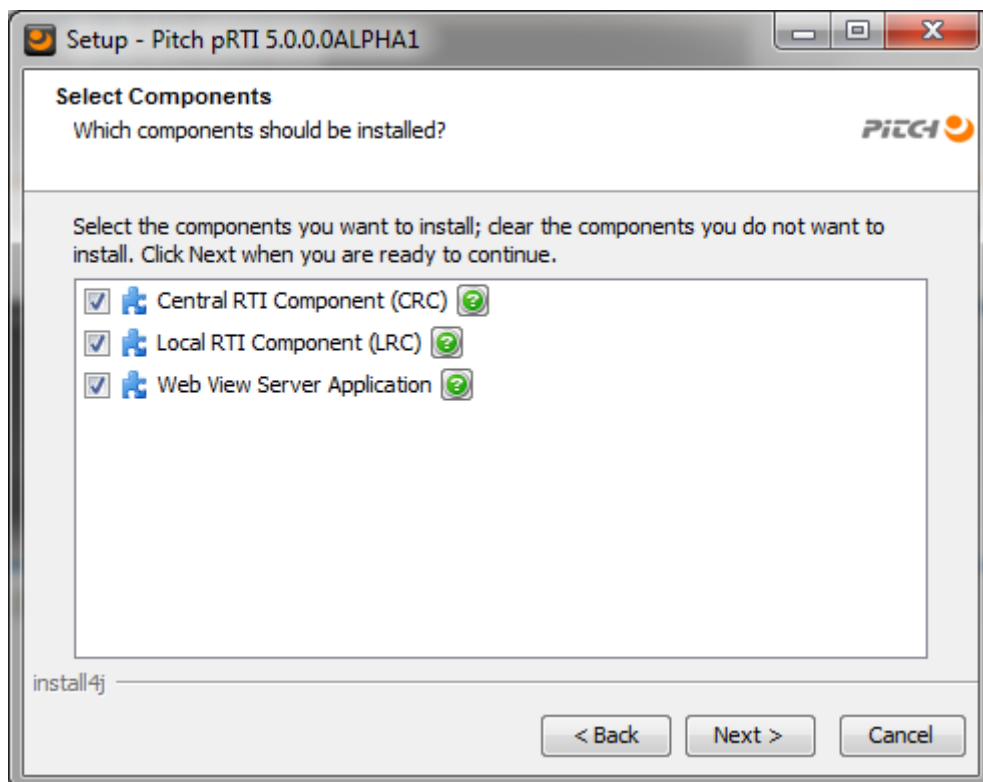


Figure 11 – Installation components selection.

Make your selection and click *Next*. You are then asked in which program group that you want the shortcuts.

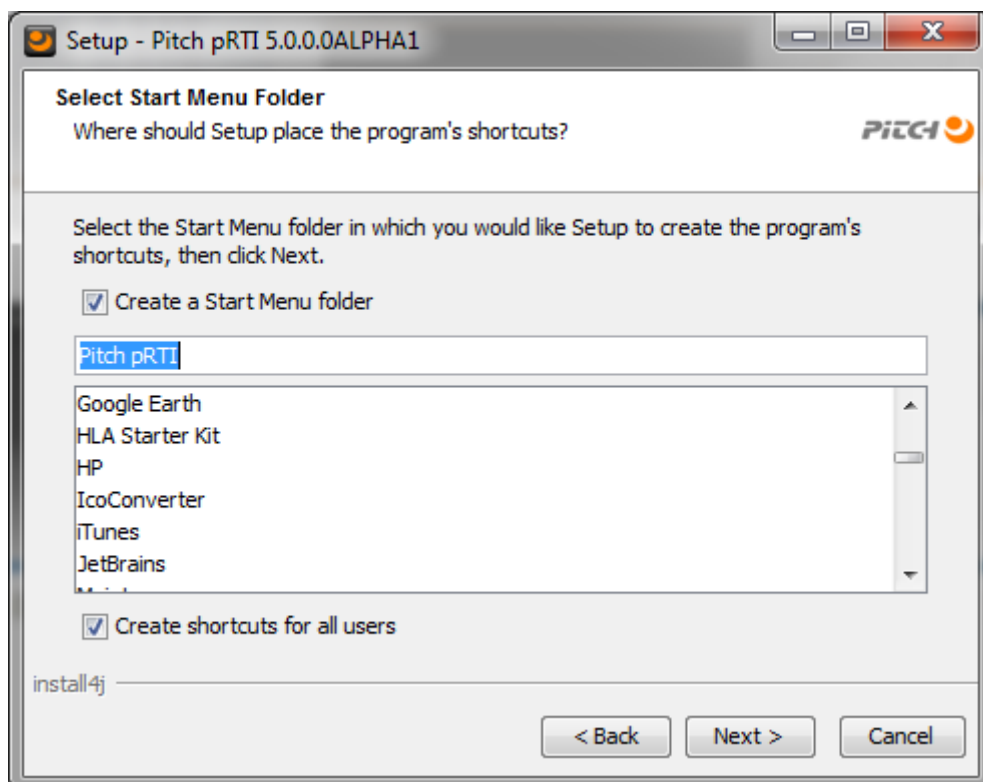


Figure 12 – Create shortcuts on the start menu.

Click *Next* to start the installation after making your selection. You are then asked to choose if you would like the installer to add the pRTI™ C++ libraries to the *PATH* variable on your system.

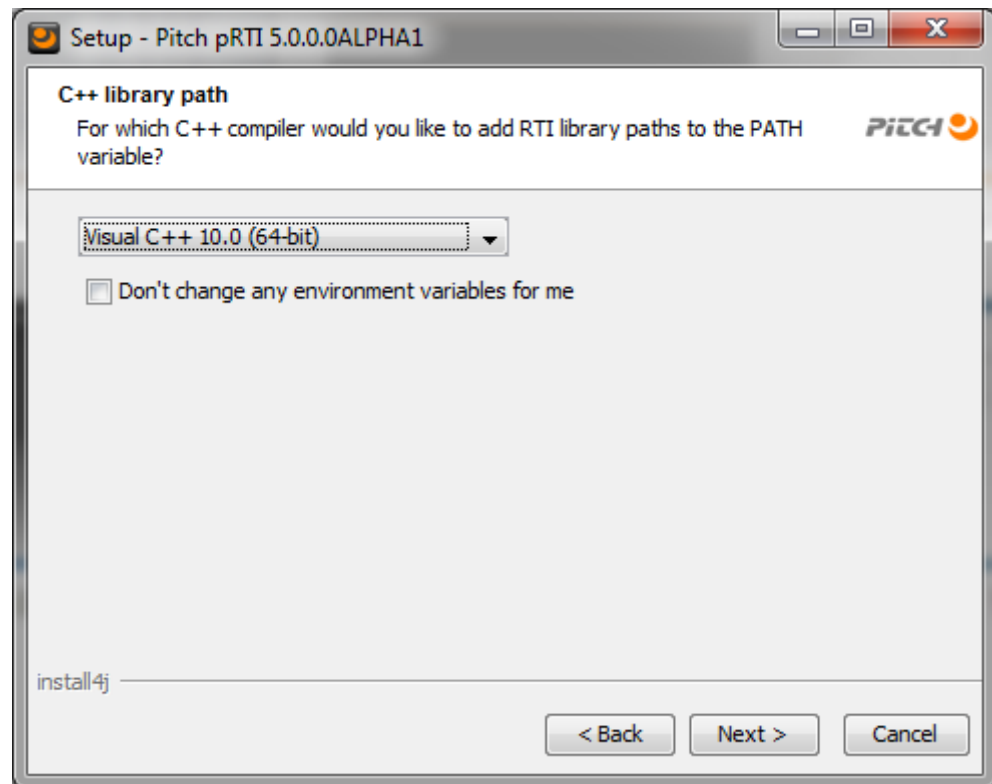


Figure 13 – C++ library PATH setting.

Click *next* to continue. You are then presented additional options such as creating desktop icons for the pRTI™ CRC and for setting an initial CRC name.

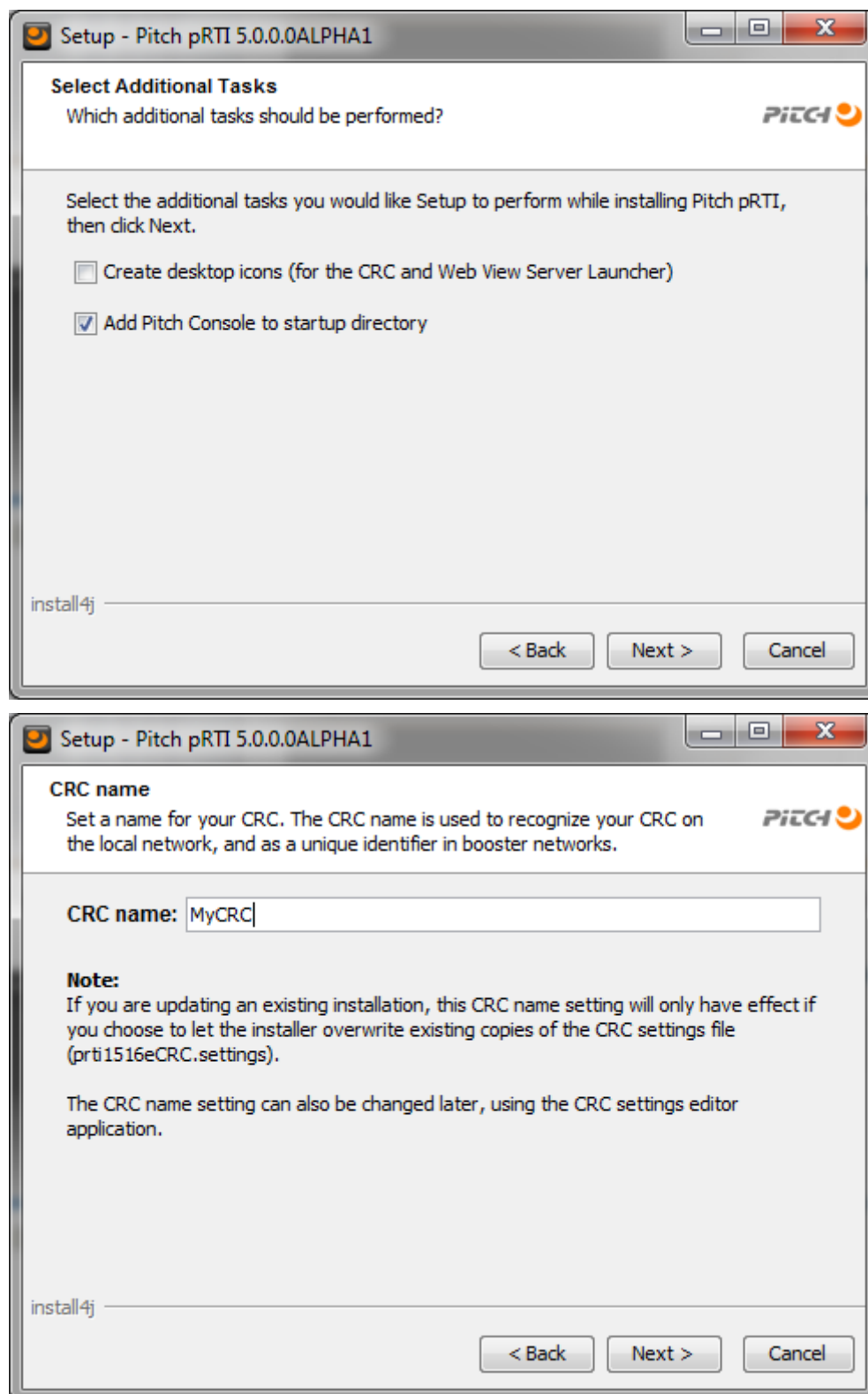
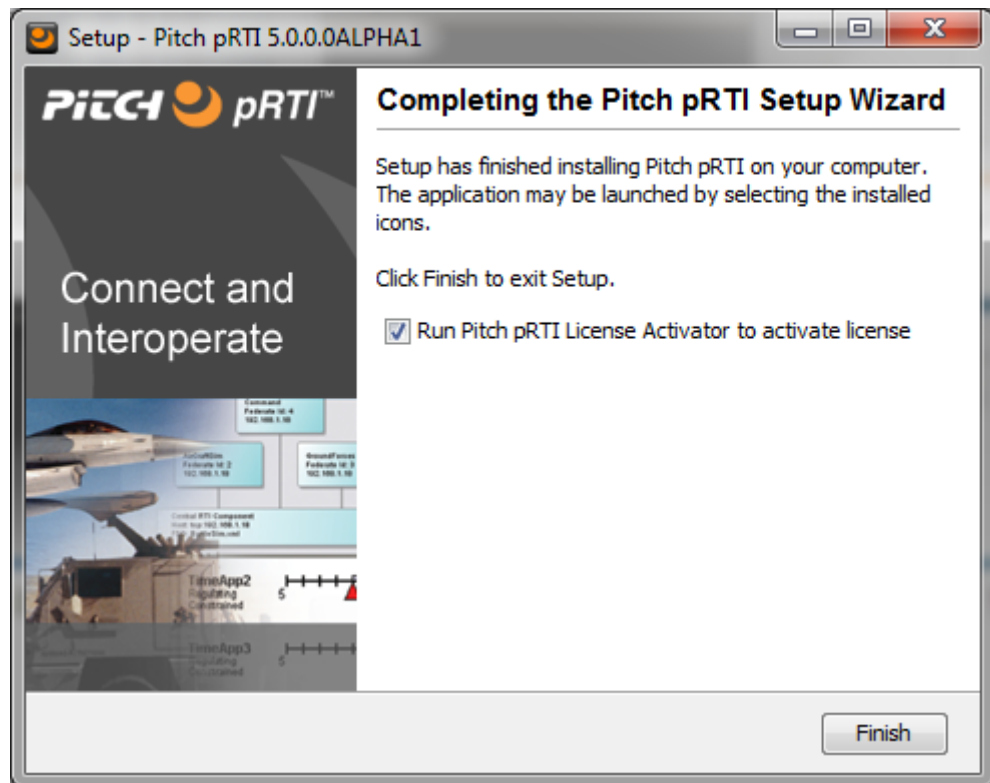


Figure 14 – Additional options.

Click *next* to finish the installation.



Click *Finish* to exit the installer.

3.2 Activating licenses

Before you can start the Pitch pRTI™ CRC, you need to activate your license with your license activation key. This is done through a separate application called *Pitch pRTI License Activator*, which is included in the installation. If you start Pitch pRTI™ without having activated the license with the License Activator, an error message will be displayed and Pitch pRTI™ will not be able to start.

Important note: In order to activate your CRC license correctly on Windows, you must run the License Activator as administrator. In order to activate your license correctly on Linux, you must run the License Activator as root. On Mac OS X the License Activator app need to be executed with writing permissions to your pRTI installation directory.

3.2.1 The license Activator Tools

The License Activator can either be executed in text mode in a command line terminal or in graphical mode. The graphical version is available in the *Pitch pRTI* folder on the Windows start menu and in the applications menu on most Linux desktop environments. The graphical version can also be started directly by executing the application *LicenseActivatorGui* in the Pitch pRTI™ installation directory. The command line version can be started the same way, by executing the application *LicenseActivator*.

IMPORTANT NOTE!

Using the command line version may require use of strong quoting (' and not ") around the license activation key.

For local federate licenses, which is a new feature described in the next subsection, the LRC Settings editor application is used for license handling.

3.2.2 License Activation Modes

Starting with v 4.5, Pitch pRTI™ come with some new modes for license handling:

Single CRC license key

This is the way that pRTI™ previously has been handling licenses, and therefore it is the most widely used mode. One single license key is used for activating the CRC, and this key allows for a certain number of federates to be used with the CRC. It also references a hardware-lock such as a USB-dongle.

Multiple CRC license keys

In this mode, one CRC license key is being used as the primary license key which determines if the CRC is allowed to start or not and also contains an allowance for a number of federates. This is handled as the single CRC license key above.

In addition to the primary license key, a number of additional license keys can be activated for the CRC. The number of federates allowed in each of the additional keys, are added to the total amount of federates allowed on the CRC – as long as the additional keys are valid and their hardware lock (such as USB-dongle) can be detected.

Local federate license key

In this mode, the license key is handled by the LRC instead of the CRC. The federate running the LRC is bringing its own license to the CRC, instead of consuming one of the federate allowances on the CRC license. Note that the CRC will still need a primary license of its own to start up.

The local federate license can hold one federate allowance for the federate itself, but it can also contain a number of additional federate allowances to be used by other federates. So, if a LRC is bringing a local federate license allowing 5 federates, it will enable the use of additional 4 other federates on that CRC. When the LRC is bringing these license allowances to the CRC it can be done with two types of restrictions:

Restricted - this means that additional allowances may only be used by other federates which has brought the same license key to the CRC.

Public - this means that the additional licenses brought by the federate can be used by any other federate, no matter if and which license key it has brought.

NOTE – Federates using the local license need to have <prti-install-dir>/lib on their library path. This is controlled using the Java system property **java.library.path**. The C++ chat sample applications has a file named *prti.vmoptions* which shows how to set this property for C++ federates. The same line, -Djava.library.path=<prti-install-dir>/lib should be added to the command line of Java federates using the local federate license.

Floating license key

An alternative to using a local CRC license key is to connect to a Pitch Floating License Server™. To configure pRTI to use a floating license, open the CRC Settings tool, click the license tab and enter the address of the Pitch Floating License

Server™. The federate count field is used to configure how many federates that shall be allocated when requesting a license lease.

3.3 Verifying the Windows Installation

It is now time to verify your installation.

Start Pitch pRTI™ using the start menu shortcut:

Start → Programs → Pitch Pitch pRTI → Pitch pRTI

This starts Pitch pRTI™ with the graphical user interface and no command prompt, so in case you want to use the command line interface you may instead start Pitch pRTI™ with both a graphical user interface and a command prompt through a different start menu shortcut:

Start → Programs → Pitch pRTI → Pitch pRTI (with console)

If your license has not been activated, you will be asked to run the License Activator tool first.

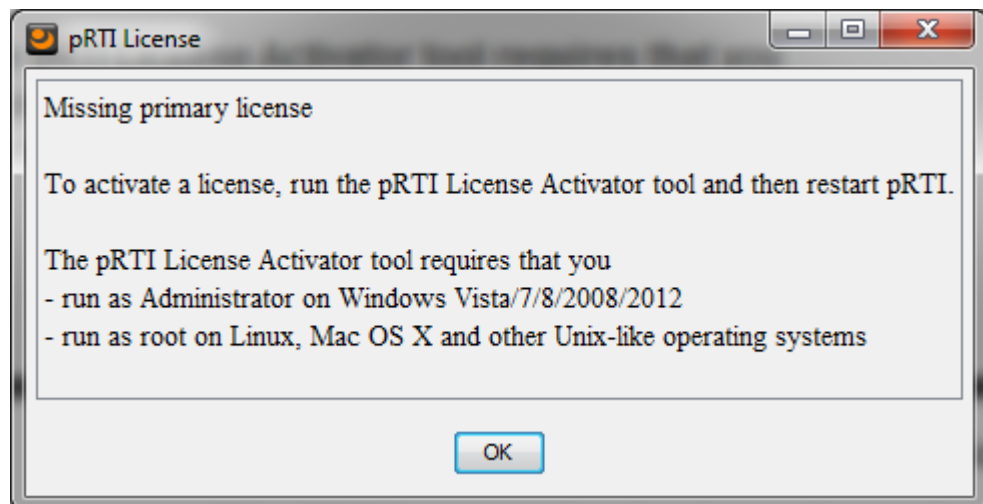


Figure 15 – License activation notification.

Important note: The CRC license activation is used to activate the Central RTI Component license. You are only allowed to use the license number on one computer.

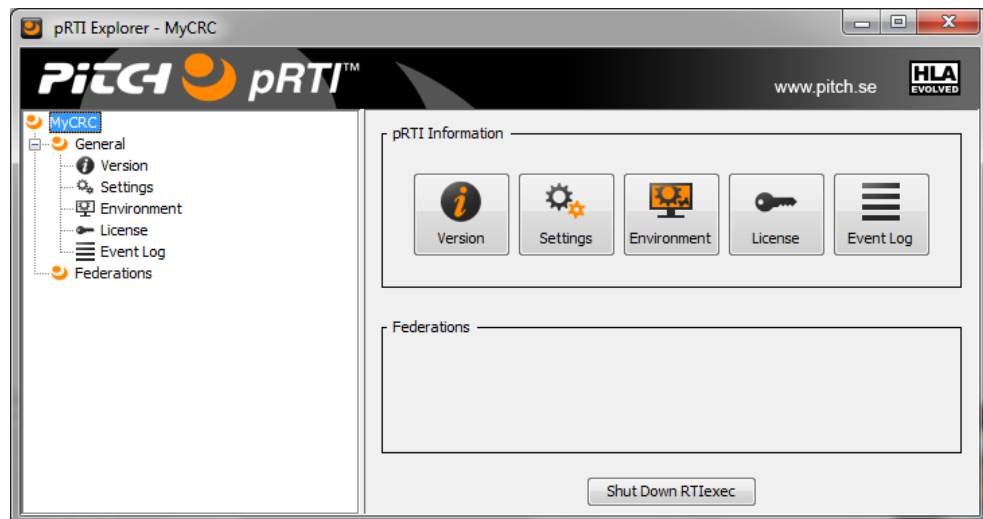


Figure 16 – The Pitch pRTI™ Explorer.

Start a Java based federate using the start menu command:

Start → Programs → Pitch pRTI → Samples → Chat Federate (Java)

Start a C++ based federate using the start menu command:

Start → Programs → Pitch pRTI → Samples → Chat Federate (Visual C++ 9.0)

You will be prompted to enter the IP-address of the CRC host, press enter to choose localhost.

Enter the IP address of the CRC host [localhost]:

You will be prompted to enter a name, as an example we will use Fred and Barney.

Enter your name: Fred

After entering the names, type a message in one of the federate windows and press the *Enter* key.

> Hello Barney

The message will now appear in the other federate window (Barney's) as:

> Fred: Hello Barney

Now click in the other federate window (Barney's) and type the following and finish by pressing the *Enter* key.

> Hello Fred

The message will now appear in the other federate window (Fred's) as:

> Barney: Hello Fred

Now type a *period* (.) in each federate window followed by the *Enter* key twice to shut down the chat federates.

Switch to the Pitch pRTI™ textual interface and type QUIT followed by the *Enter* key or click *Shut Down RTIexec* in the graphical interface. Both these alternatives will shut down the CRC.

If you want to run the chat sample over a network, simply enter the IP-address of the CRC host when prompted by the chat program instead of using localhost. Watch it connect to the remote CRC of the pRTI™.

3.4 Linux Installation

Before you start, check that you have

- The installation executable.
- Your license activation key that you will use to activate the software, in case of installing a CRC or a locally licensed LRC.

Execute the installer file from a terminal window. Note that you may have to modify the file permissions using the *chmod* command before running the file:

```
[root@Enorm root] # ./chmod u+x install_prti1516_v5_Linux.sh
[root@Enorm root] # ./install_prti1516_v5_Linux.sh
```

A graphical installer will now start. The graphical installer is very similar to the Windows installer, so check the instructions in section 3.1 if anything is unclear.

3.5 Verifying the Linux Installation

In the bin directory, found in the installation directory, there is an executable, pRTI1516e, that can be used to start pRTI™. There are also shell scripts in the directories *samples/chat* and *samples/chatcc* that can be used to start Chat federates that are supplied with the default installation. There are also shell scripts and binary launchers in the bin directory, which can start editors for LRC settings, CRC settings and Trace settings. See section 16.2 for more information. To start Pitch pRTI™, run the script *prti1516.sh* if you want to have both the graphical- and command line interface or the binary launcher *pRTI1516e* if you only want to use the graphical interface of the CRC. If you are using the full version of Pitch pRTI™ (not LE) you will be required to enter the license activation key using the *LicenseActivator* application in case your license has not yet been activated.

After activating your license you will be able to start the CRC graphical user interface, called pRTI Explorer, as shown in Figure 17.

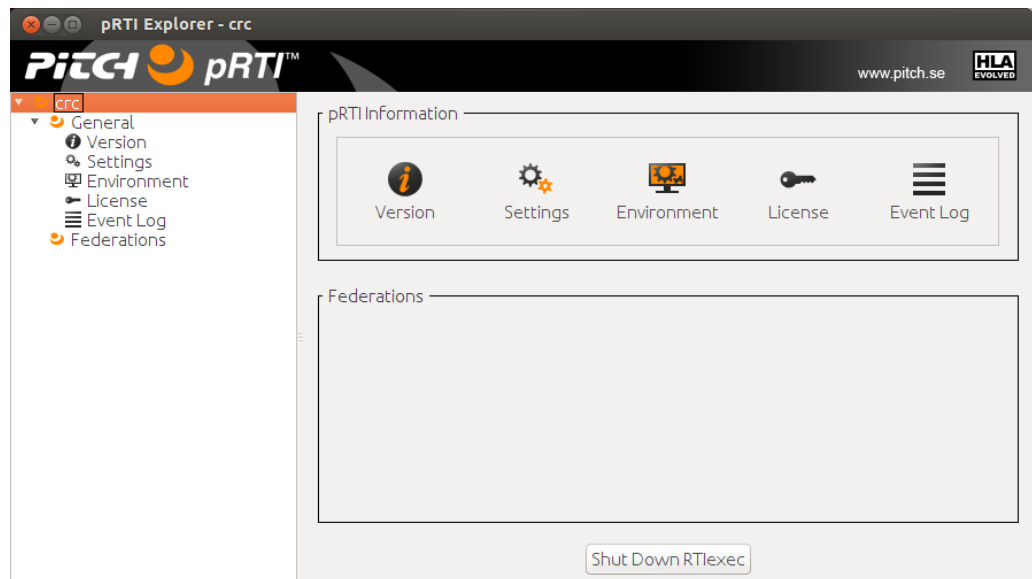


Figure 17 – The pRTI™ Explorer

Important note: The license activation key is used to activate the Central RTI Component. You are only allowed to use the license activation key on one computer.

To start the Java Chat federate, run the *chat-java-1516e.sh* file in a new command prompt window. Make sure that pRTI™ is running before you start the federate.

```
[root@Enorm root]# cd /opt/prti1516e/samples/chat-java/
[root@Enorm bin]$ ./chat-java-1516e.sh
Enter the address of the CRC host [localhost]: <ENTER>
Enter your name: Fred
Type messages you want to send. To exit, type . <ENTER>
>
```

Now, to start the C++ Chat federate, open another command prompt window and run the *chat-cpp-1516e_gccXX.sh* file.

```
[root@Enorm root]# cd /opt/prti1516e/samples/chat-cpp/
[root@Enorm bin]# ./chat-cpp-1516e_gcc41.sh
Enter the address of the CRC host [localhost]: <ENTER>
Enter your name: Barney
Type messages you want to send. To exit, type . <ENTER>
>
```

Enter a message in one of the federate windows (Barney's):

```
> Hello Fred
```

Finish by pressing the *Return* key. Watch the message appear in the window of the other federate:

```
> Barney: Hello Fred
```

Now click in the other federate window (Fred's) and type:

```
> Hello Barney
```

Finish by pressing the *Return* key. Watch the message appear in the window of the other federate:

```
> Fred: Hello Barney
```

Now type a *period* (.) in each federate window followed by the *Return* key twice.

Switch to the pRTI™ command line interface and type `QUIT` followed by *Return* or press the button *Shut Down RTIexec* in the pRTI Explorer window to shut down Pitch pRTI™.

3.6 Installing on Other Platforms

Installing Pitch pRTI™ on other platforms than those described above, is made in a similar way unless a separate instruction is provided.

4 Uninstalling Pitch pRTI™

4.1 Uninstalling on Windows

To uninstall Pitch pRTI™ on Windows simply use the *Add or Remove Programs* application available in the *Control Panel*.

This will start the graphical uninstaller which will guide you through the uninstall process.

4.2 Uninstalling on Linux

To uninstall Pitch pRTI™ on Linux run the *uninstall* executable found in the root of the Pitch pRTI™ installation.

```
[root@Enorm root]# sudo /opt/prti1516e/uninstall
```

This will start the graphical uninstaller which will guide you through uninstall process.

Note: If you have installed Pitch pRTI™ as a daemon please uninstall the daemon prior to uninstalling Pitch pRTI™.

4.3 Uninstalling on Other Platforms

Uninstalling Pitch pRTI™ on other platforms than those described above, is done in a similar way unless a separate installation instruction provided for that platform.

5 Running Pitch pRTI™

This section describes how to run Pitch pRTI™ and what can be done using the graphical user interfaces and the command line interface.

Please see chapter 3 for information about how to start Pitch pRTI™.

5.1 Graphical User Interfaces Overview

- **The CRC GUI, pRTI Explorer**, which is available on the computer with the CRC. It focuses on the management of the federations and federation-wide use of HLA services. It enables you to monitor how federates join, resign, publish, subscribe, register objects etc. These features are also available through Pitch Web View™. There is also a graphical user interface for doing CRC settings named *CRC Settings*.
- **Pitch Control Center**, which is a graphical user interface available on each computer running a federate. It focuses on local aspects of each federate, such as the connection status and performance during execution. Pitch Control Center is described in more detail in section 8. In addition to this, there is a graphical user interface for LRC settings, which can be used for configuring the LRC before the execution starts.

If you have a federate running on the same computer as the CRC, you can use both pRTI Explorer and Pitch Control Center.

5.2 Using the Graphical User Interface

The graphical user interface lets you monitor, inspect and debug HLA federations. The main advantages of the graphical user interface are:

- Easier to get started with HLA for new developers.
- Improved productivity when developing and debugging federates.
- Improved productivity during integration and testing of federations.
- Improved monitoring during execution of simulations.

The main functionality includes:

- Inspect and modify the configuration of the CRC.
- Inspect the lifecycle of federations and federates.
- Inspect the FOM at runtime.
- Resign and destroy federates and federations.
- Check synchronization points.
- List central information about registered objects and attribute ownership.
- Monitor time management information for federates graphically.
- Inspect communication links between federates and the CRC.

- Inspect publications and subscriptions for each LRC.
- Inspect which objects are discovered as which class for each LRC.
- Control tracing of RTI calls and callbacks for each LRC.
- And more...

When Pitch pRTI™ is started, both the command line interface and the graphical user interface can be started. Note if a CRC is already running on the computer the dialog box shown in Figure 18 will be displayed.

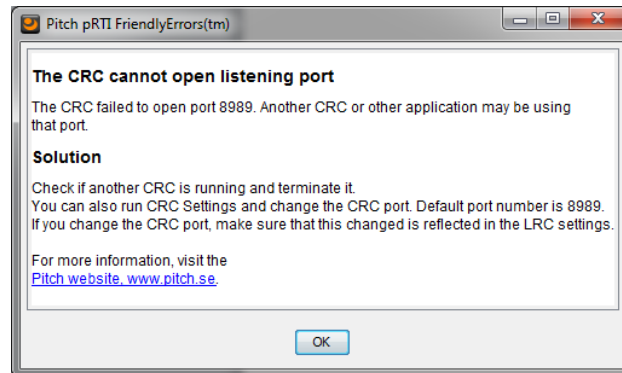


Figure 18 – Error message when multiple CRC instances are started.

Either shut down the other CRC or specify a different port as described in section 16.1. If you specify a different port, remember to make sure that all the federates use this port and not the default when connecting to the CRC.

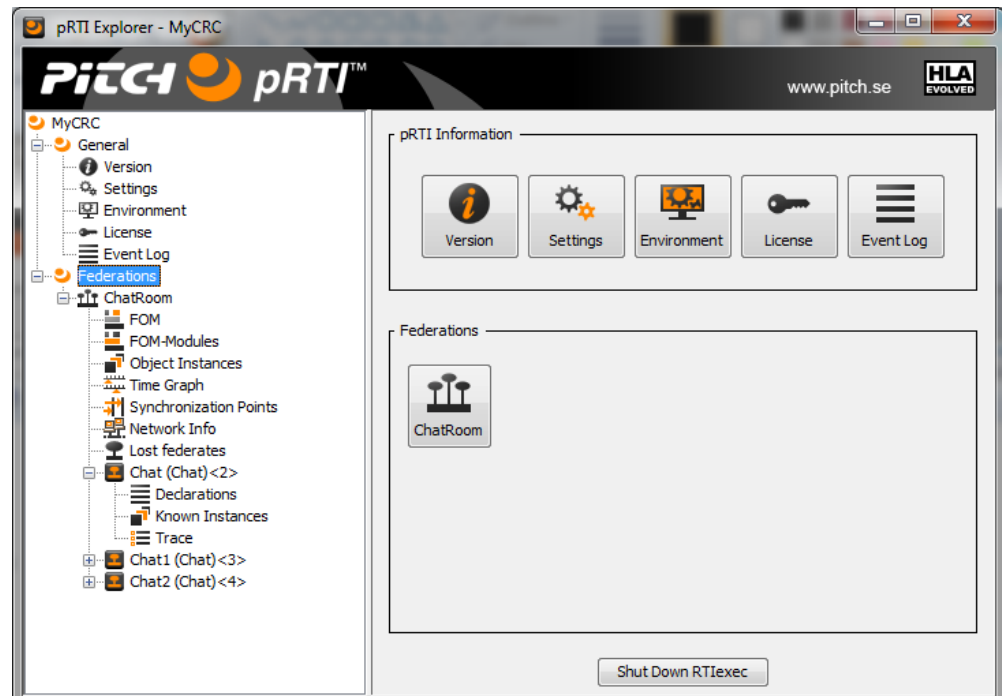


Figure 19 – Pitch pRTI™ start panel.

Figure 19 shows the graphical user interface. To the left is a tree graph where you can navigate by expanding and collapsing branches. To the right is a panel displaying information corresponding to the current selection in the tree.

The *General* node contains information regarding the CRC such as version, settings, run-time environment and license, while the *Federations* node contains information about the federation executions that are currently running. The following sections provide a brief overview the main views available under these two nodes.

5.2.1 Version

Figure 20 shows the version panel which contains a *Check for Updates...* button. Clicking this will open your default web browser and guide you to a webpage which will show you if there are any Pitch pRTI™ updates available.

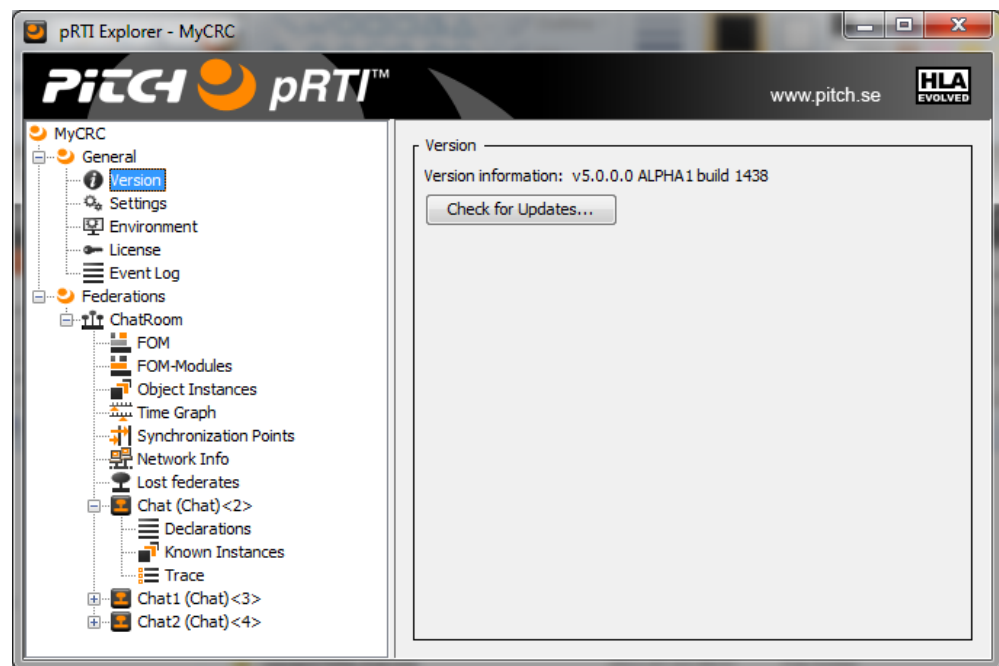


Figure 20 – The version panel.

5.2.2 Settings

Figure 21 shows the settings panel. For more information on this see section 16.1.

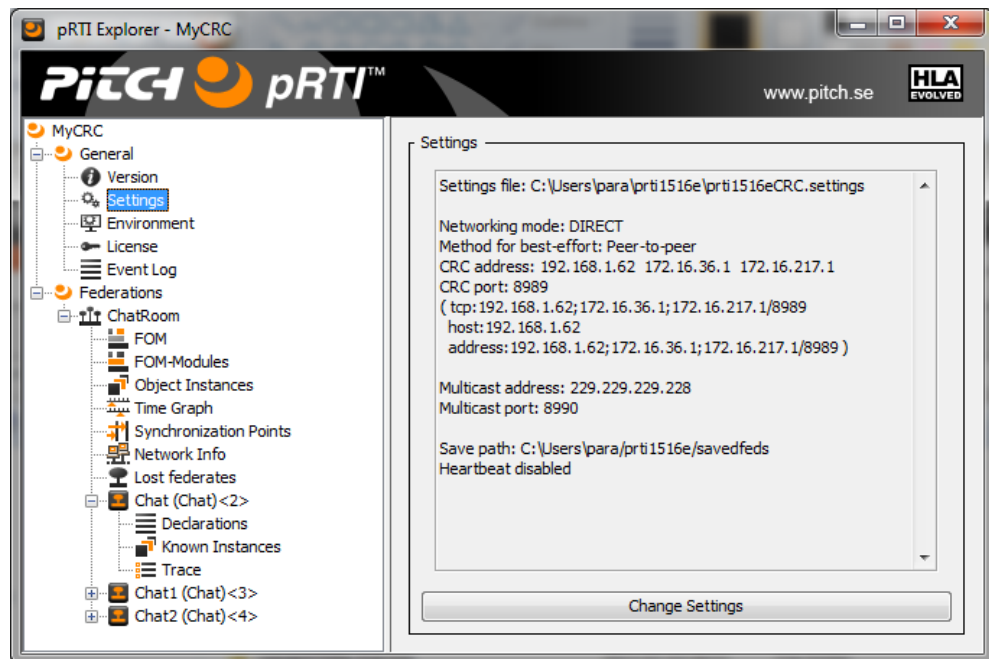


Figure 21 The settings panel.

5.2.3 Environment

Figure 22 shows the environment panel. This provides information on which JRE is being used and other environmental data.

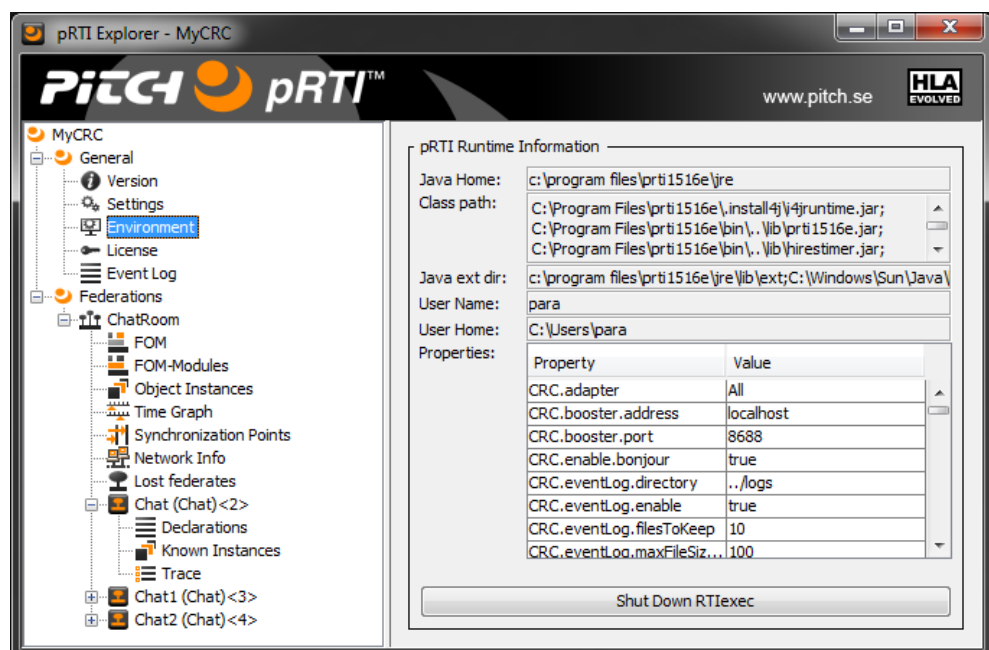


Figure 22 – The environment panel.

5.2.4 License

Figure 23 shows the license panel which provides information on the current license being used. Here you can see which functionality you license includes and also when it will expire if it is not a permanent license. Entering a new license can however only be done using the License Activator application.

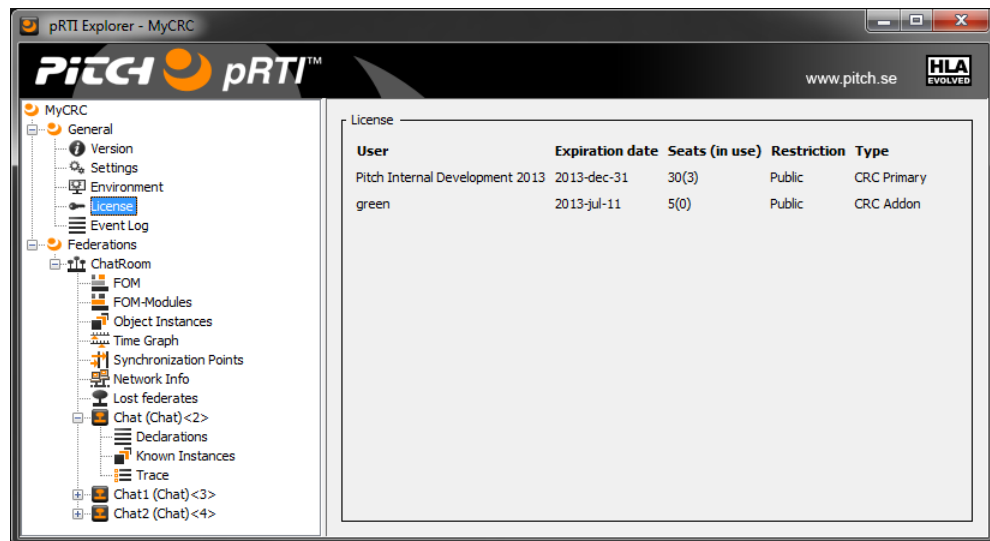


Figure 23 – The license panel.

5.2.5 Event Log

Figure 24 shows the event log panel. This event log is similar to that found on the Windows operating system. It maintains a log of events that occurs, e.g. the creation of a federation execution, crashing federates etc.

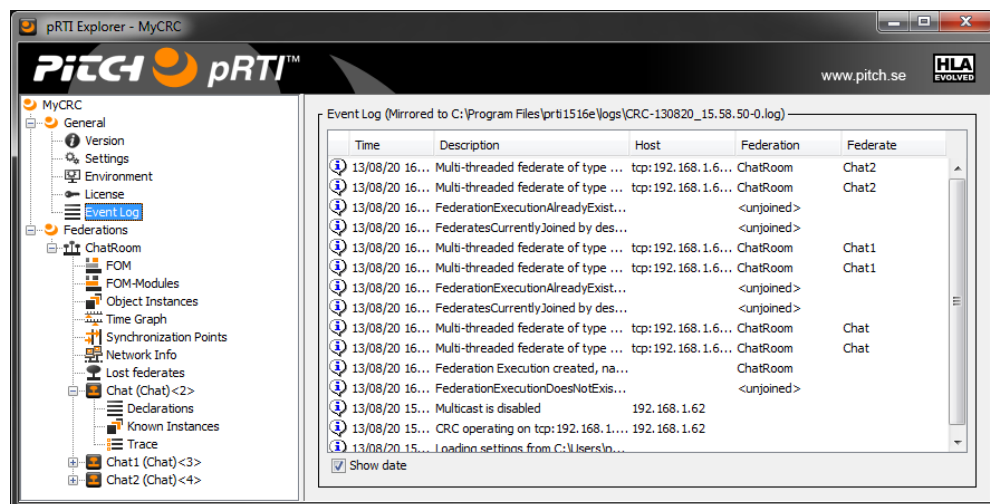


Figure 24 – The event log panel.

5.2.6 Federation Overview

Clicking on the *ChatRoom* federation icon brings you to the view shown in Figure 25. This shows information about the federation, including information about the Central RTI Component and each participating federate.

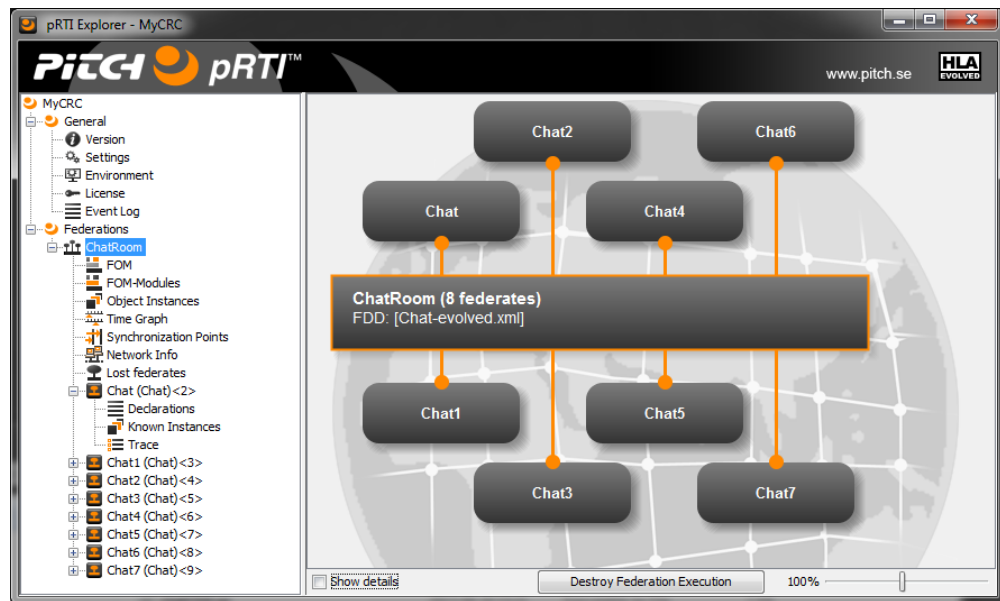


Figure 25 – The federation lollipop view.

5.2.7 Federation Object Model (FOM)

The FOM view lets you inspect the FOM used in the selected federation at runtime. Figure 26 illustrates how the FOM view displays the parameters of the *Communication* interaction class.

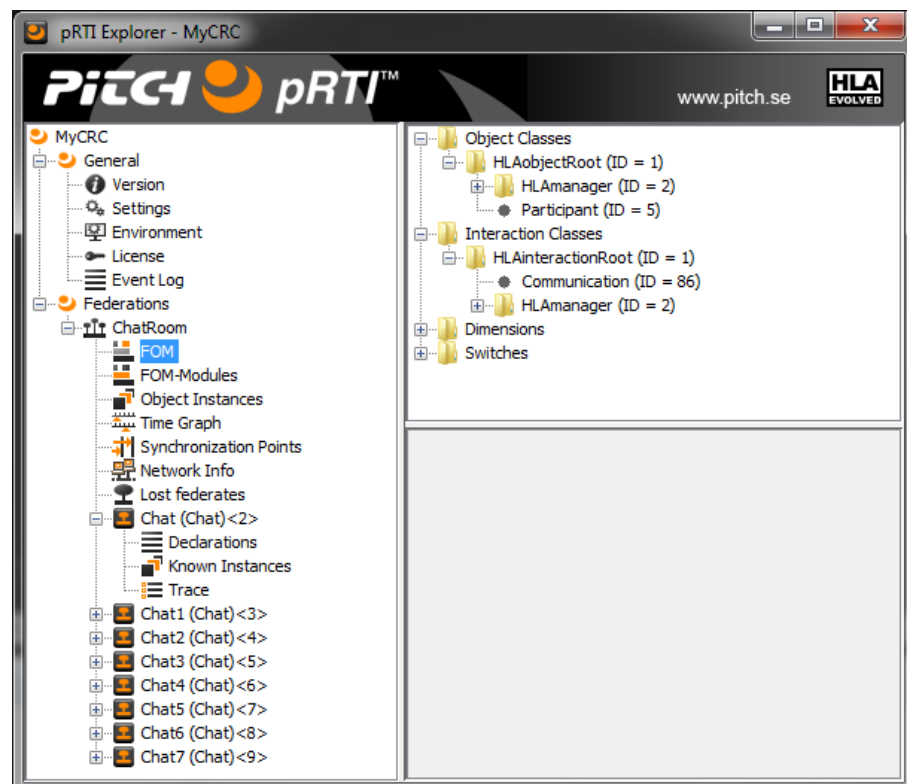
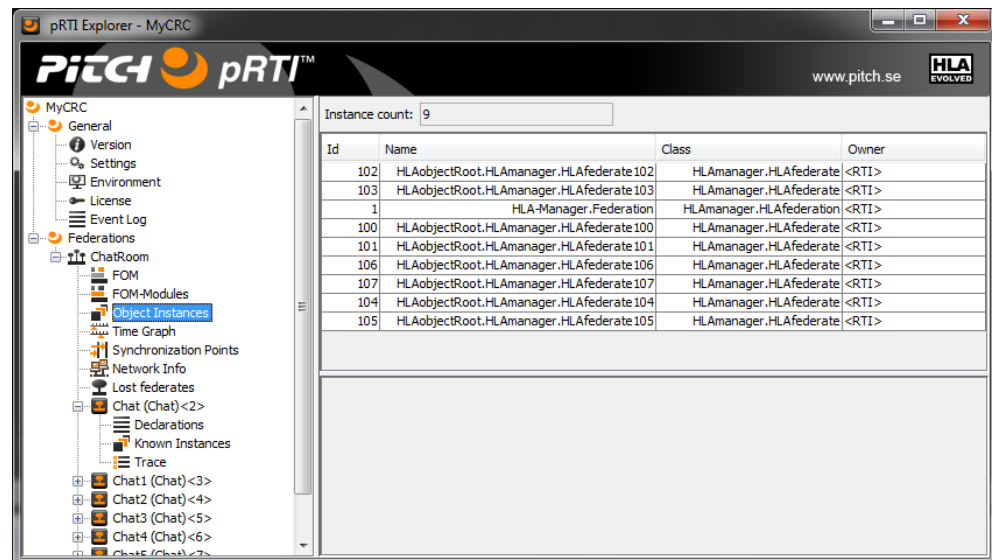


Figure 26 – Inspection of the FOM.

5.2.8 Object Instances

The view shown in Figure 27 gives a detailed view of all the object instances that are currently registered in the federation. For each object instance, all the attributes can be listed by clicking on the object instance in the table.



Instance count: 9

Id	Name	Class	Owner
102	HLAobjectRoot.HLAManager.HLAfederate102	HLAManager.HLAfederate	<RTI>
103	HLAobjectRoot.HLAManager.HLAfederate103	HLAManager.HLAfederate	<RTI>
1	HLA-Manager.Federation	HLAManager.HLAfederate	<RTI>
100	HLAobjectRoot.HLAManager.HLAfederate100	HLAManager.HLAfederate	<RTI>
101	HLAobjectRoot.HLAManager.HLAfederate101	HLAManager.HLAfederate	<RTI>
106	HLAobjectRoot.HLAManager.HLAfederate106	HLAManager.HLAfederate	<RTI>
107	HLAobjectRoot.HLAManager.HLAfederate107	HLAManager.HLAfederate	<RTI>
104	HLAobjectRoot.HLAManager.HLAfederate104	HLAManager.HLAfederate	<RTI>
105	HLAobjectRoot.HLAManager.HLAfederate105	HLAManager.HLAfederate	<RTI>

Figure 27 – Inspection of object instances and attributes.

5.2.9 Time Graph

Figure 28 shows the time management state for the entire federation. The current time value for each federate is illustrated by a red triangle. The lookahead is shown as a blue rectangle. The grey area represents a time value that the federate is not allowed to achieve yet, provided that the federate is time constrained. The vertical black indicator shows the minimum time-stamp that the federate will attach to any new messages.

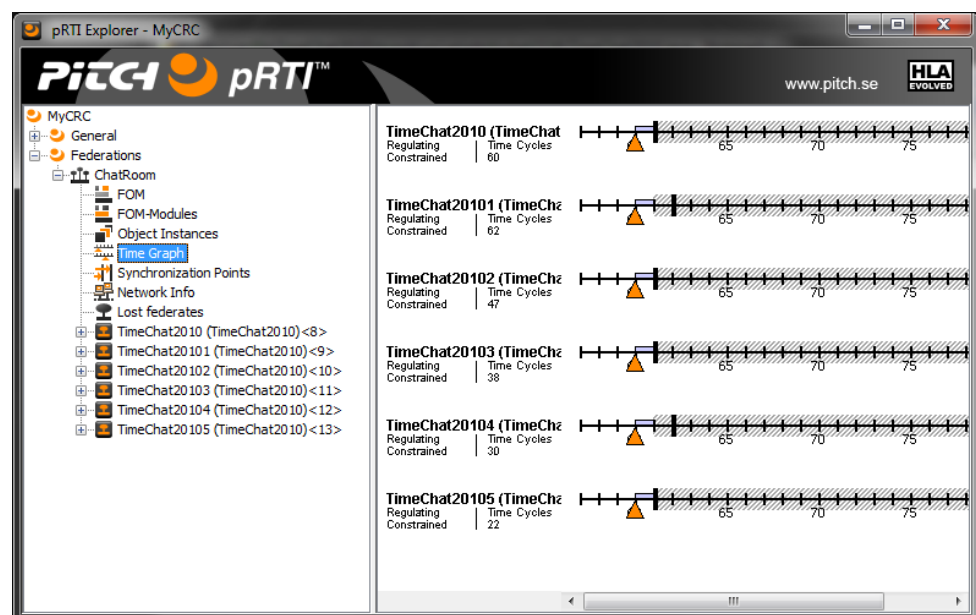


Figure 28 – The time status for each federate.

5.2.10 Synchronization Points

Figure 29 shows the synchronization points that exist in the federation. This view will be empty if no synchronization points have been registered. Figure 29 shows three different synchronization points, and also the status for each federate in the federation. The synchronization point labeled *Initialized* has been achieved by the federate with federate id 3. The federates that are listed in the *Pending Federates Id* column have not achieved the synchronization point yet.

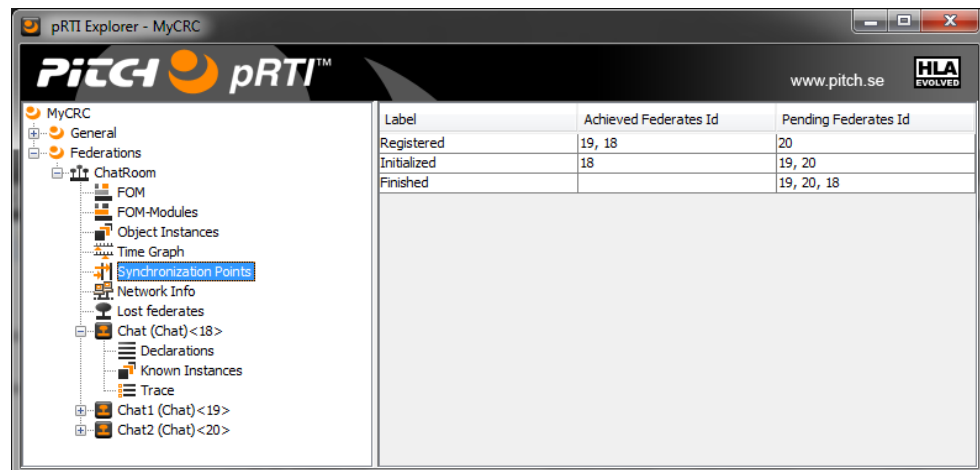


Figure 29 – The synchronization points and federate synchronization status.

5.2.11 Network Info

The Network info view lets you inspect the network links between the CRC and the LRC:s in the federation. Clicking on a specific connection shows you statistics for that connections network traffic as shown in Figure 30.

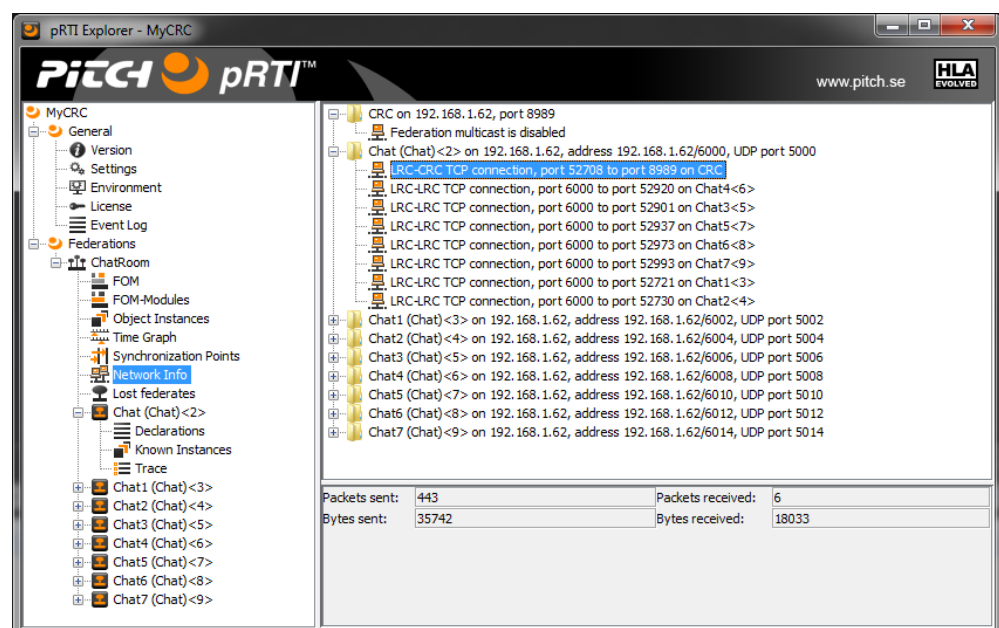


Figure 30 – The network information for federation participants.

5.2.12 Federate Information and Tracing

The object instances, time graph, synchronization points and the network information are details pertaining to the federation execution. The GUI also gives you the possibility to view information about each federate.

Figure 31 shows some general information about the federate. The federate id, the host of the computer where the federate is executing and some time management information is displayed.

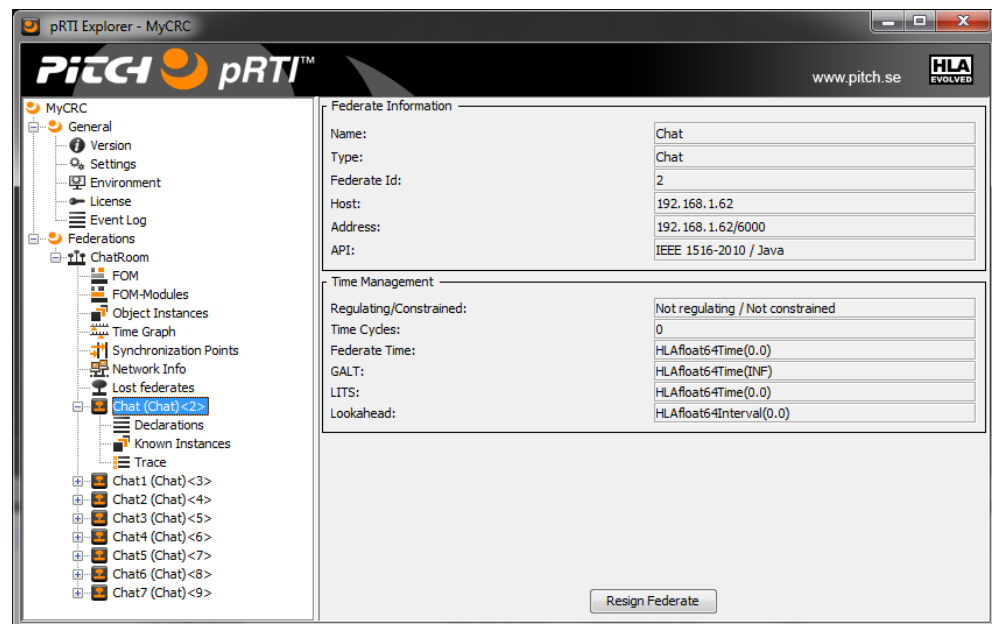


Figure 31 – The federate information panel.

Check the *Tracing enabled* checkbox to start tracing all the activity of the selected federate. Select the target of the tracing by clicking the *Trace to...* button. Figure 35 on page 41 shows how to select specific services that should be traced.

It is also possible to resign the federate from the federation execution.

Figure 32 shows the declarations made by the federate. Select *Declarations* for a federate in the tree view and clicking the buttons displays information as described below:

- *User Object Classes* - displays the object class declarations made by the federate.
- *User Interactions Classes* – displays the interaction class declarations made by the federate.
- *MOM Object Classes* – displays MOM object class declarations made by the federate.
- *MOM Interaction Classes* - displays MOM interaction class made by the federate including.

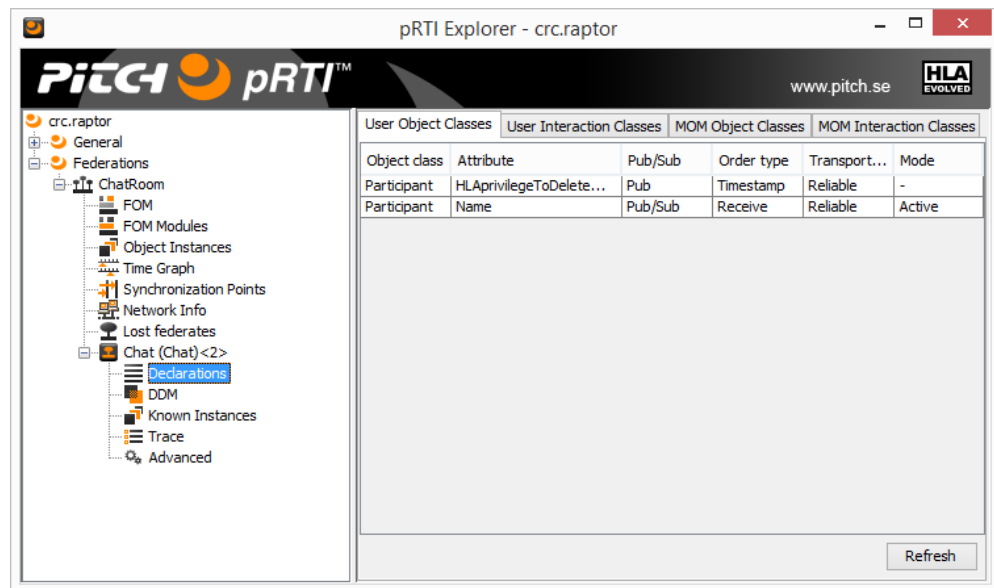


Figure 32 – The object and interaction class declarations.

Figure 33 shows DDM information for the federate:

- *Object Class Subs* – displays subscribed object classes
- *Object Class Subs By Region* – displays subscribed object class attributes and the DDM region subscription
- *Attribute Associations* – displays attribute associations with regions
- *Regions* – lists the regions created by this federate

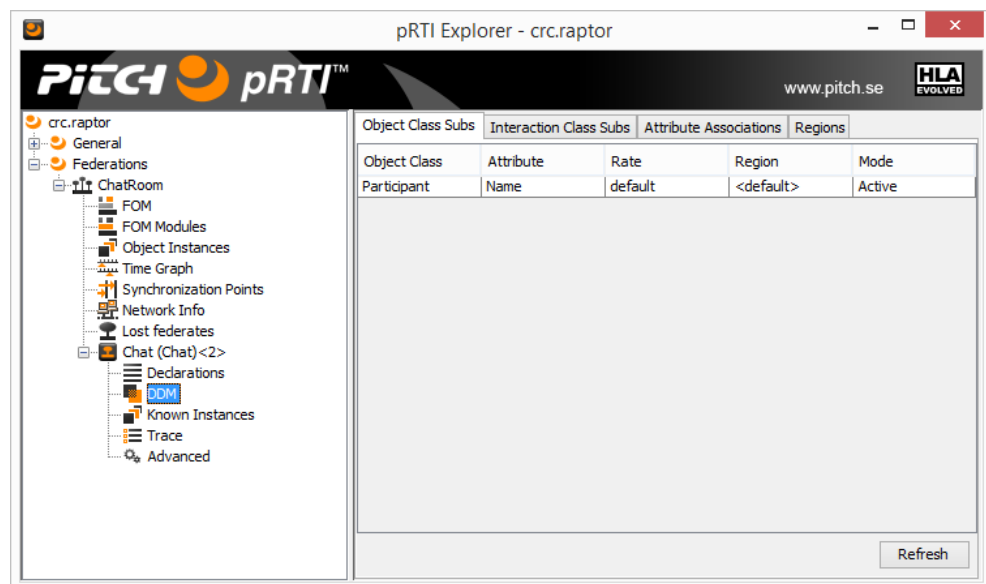


Figure 33 – The DDM information.

Figure 34 shows the object instances that have been discovered by the federate. The *Class* column shows the class that the object was registered as (i.e. the object class used by the federate who registered the object). The *Known Class* column shows the class that the federate has discovered the object as.

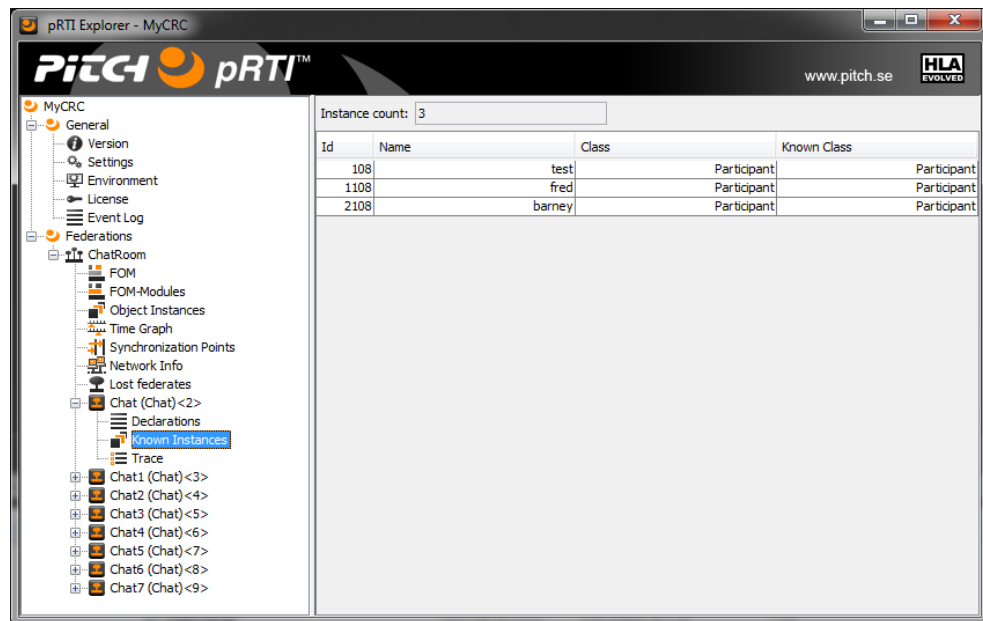


Figure 34 – The discovered object instances.

Figure 35 shows an example of the tracing possibilities. By selecting either an entire service group (such as Declaration Management) or by selecting an individual service, Pitch pRTI™ logs all the calls and callbacks belonging to that service or service group. There is also a separate trace settings editor, named *Trace Settings* which can be used for doing trace settings prior to federation startup. This application can be found in the start menu on Windows and in the *bin* directory of the Pitch pRTI™ installation.

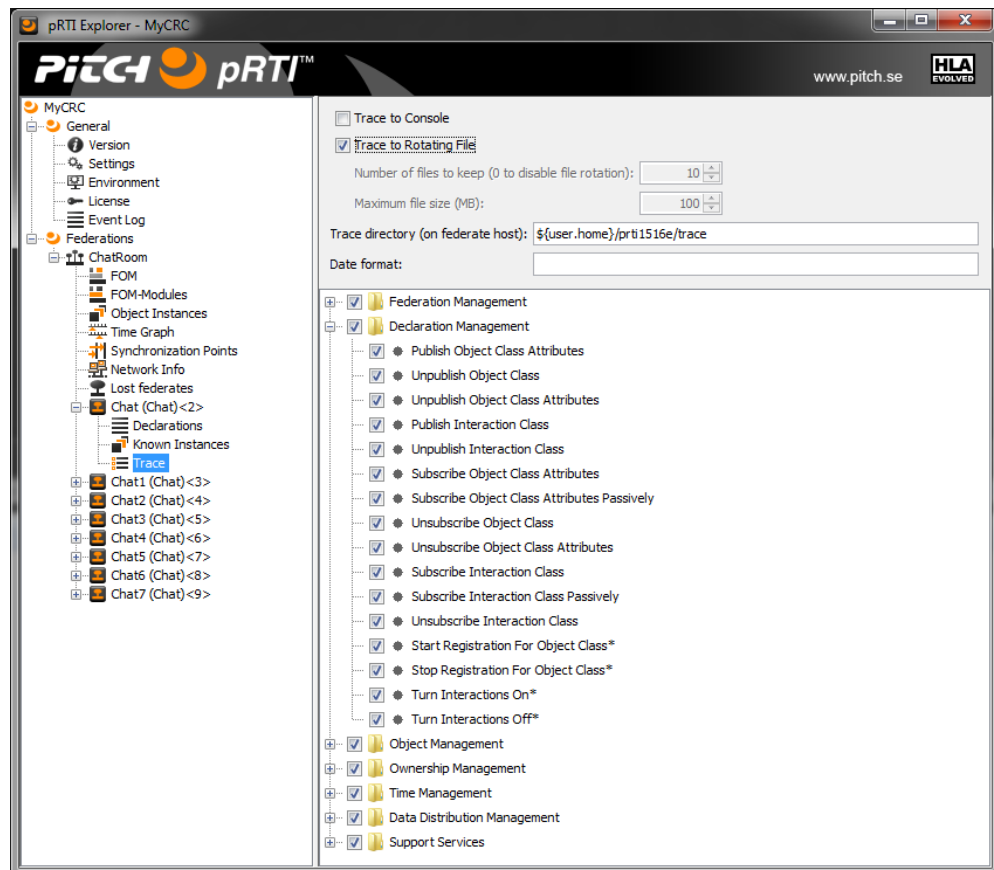


Figure 35 – The tracing panel.

Please note that the log file grows quite rapidly if you have a federation with many frequent updates. Performance for your federation is also affected in a negative way if tracing is enabled.

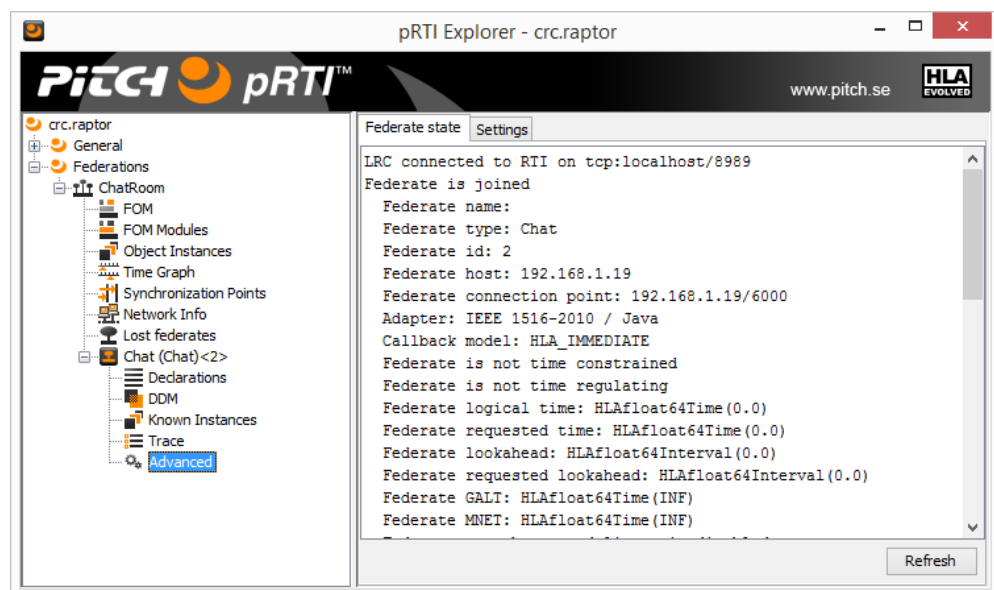


Figure 36 – The advanced panel.

Figure 36 shows the advanced view where LRC diagnostics information as well as effective LRC settings can be inspected.

5.3 Using the command line interface

Pitch pRTI™ can be started with or without a command line user interface. On Windows start menu, there are two starting options of which one starts Pitch pRTI™ with a command line user interface. Starting Pitch pRTI™ with the command line user interface on other platforms is done by starting it through the shell script available in the bin directory of the Pitch pRTI™ installation.

The command line provides the following commands:

Command	Description
QUIT	Quits pRTI™ 1516.
HELP	Lists the available commands.
GUI	Starts a graphical interface.
VERBOSE [<file>]	Turn on verbose logging to the specified file. If no file is specified, the logging is done to the screen.
SILENT	Turns off verbose logging.
DUMP	Prints the internal data of the RTI executive. Also useful for displaying the TCP/IP addresses of all joined federates.
DUMP <federation name> <federate id>	Prints the internal data of the specified federate.
LIST	Lists all joined federates.
RESIGN <federation name> <federate id>	Resigns the specified federate.
DESTROY <federation name>	Destroys the specified federation.
LICENSE	Prompts for a new license key.
LICENSE SHOW	Prints the current license information.
VERSION	Prints version information in RTIexec and every federate.
ENV	Prints CLASSPATH and Java environment information.
SETTINGS	Prints the current CRC settings.

6 Running Pitch pRTI™ In Service-Mode

6.1 Introduction

It is possible to run Pitch pRTI™ CRC as a background service on Windows and as a daemon on Mac OS X, Linux and unix platforms.

The benefits of running the CRC as a service include:

- Ease of use - the CRC service is managed along with other services on the computer.
- There is no need for a user to be logged in while the CRC is running.

6.2 Limitations

When the CRC service is started on Windows, the pRTI™ Explorer window, shown in Figure 37, is not displayed.

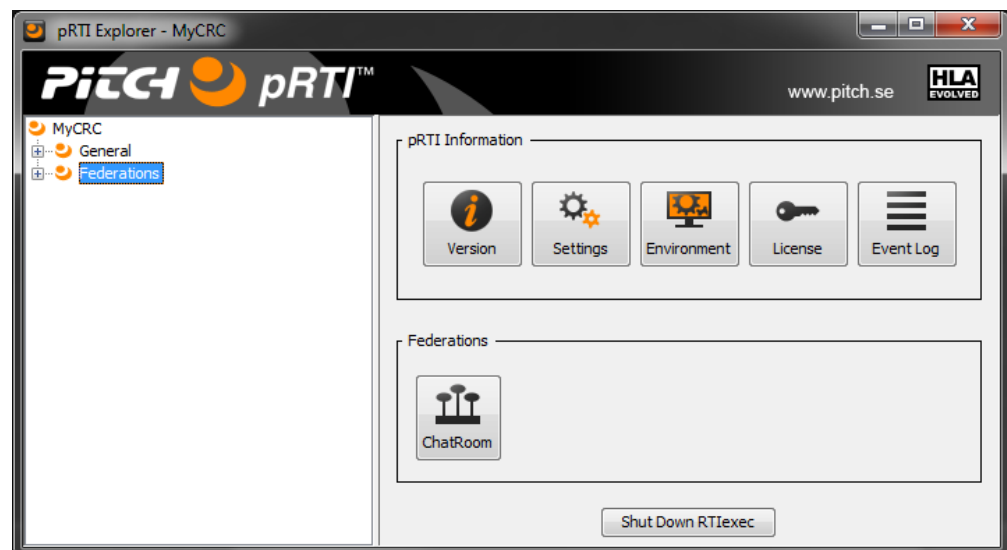


Figure 37 – The pRTI™ Explorer.

6.3 Installing the Service

When Pitch pRTI™ is installed, the components required to run it as a service are also installed by default, but a manual start-up is required to activate the service after installation. By default the service is not set to be started automatically.

6.3.1 Service Management on Windows

On Windows platforms, the service can be started, restarted and stopped using these three commands from the start menu:

Start → Programs → Pitch pRTI → Service → Start pRTI Service

Start → Programs → Pitch pRTI → Service → Restart pRTI Service

Start → Programs → Pitch pRTI → Service → Stop pRTI Service

Before starting the service you need to run the License Activator application to enter the license activation key.

Important Note: When Pitch pRTI™ runs as a service it logs in using the *Local System* account. This implies that when started, Pitch pRTI™ will look for settings files (among other things) in the directory defined by the environment variable `PRTI1516E_HOME` which is set to the pRTI™ installation directory by the installer wizard.

The CRC service can also be managed through the *Local Services* tool which is located in the Windows Control Panel.

6.3.2 Service Management on Linux

The pRTI™ installer installs the *pRTI1516e-service* script which allows you to control the pRTI™ CRC daemon. This script allows you to start, stop and monitor the daemon as illustrated below:

```
[para@maggie]# sudo service pRTI1516e-service start
Starting pRTI1516e-service
[para@maggie]# sudo service pRTI1516e-service status
The daemon is running
[para@maggie]# sudo service pRTI1516e-service stop
Shutting down pRTI1516e-service
Stopped.
```

Depending on which Linux distribution is being used, it is also possible to control the daemon through a graphical user interface provided with the desktop environment.

Important Note: When Pitch pRTI™ runs as a daemon it runs as the root user and will therefore not search for the CRC settings file in the user home directory, but instead in `/etc/prti1516e`. Therefore if you installed Pitch pRTI™ logged on as a user other than root, you will need to manually copy the *prti1516e* directory from the home directory of that user to the */etc* directory. There is also an original copy of the *prti1516e* directory located in the *user.home* directory under your Pitch pRTI™ installation directory. This is meant to be copied to any directory needed.

6.3.3 Logging

The pRTI™ service is logging to the logging directory of the pRTI™ installation (`<install-dir>/logs`). The log files written by the pRTI™ service are *CRC_service_stderr.log* and *CRC_service_stdout.log*. This is a good starting point for troubleshooting in case the CRC does not seem to operate correctly.

7 Using Pitch Web View

7.1 Introduction

Starting with version 4.4, pRTI™ comes with a web based user interface which substitutes the previous applet based web user interface. The major differences between pRTI™ Web View and the previous web based user interface are:

- Pitch Web View is not based on Java applets so support for Java applets in the web browser is not required.
- Pitch Web View is only using the HTTP protocol for communication between the web browser and the web server, so no special firewall configurations are needed.
- Pitch Web View is distributed as a web application archive (WAR) and can easily be deployed on most servlet application servers, such as Apache Tomcat.

The installation of Web View comes with a small web server which can be used to launch the web application locally on your computer. In cases where Web View is to be deployed in a scalable server environment, running as a service, we recommend deploying it in an application server.



Figure 38 – Pitch pRTI™ Web View.

The benefits of the web based user interface include:

- The information provided through the pRTI™ Explorer is made accessible to all sites and computers during a distributed federation execution as opposed to only the site where the pRTI™ CRC is executing.
- Configurable security allows access control allowing the pRTI™ administrator to decide who should have access to the graphical user interface.
- Combined with running pRTI™ in service-mode as describe in chapter 6, the Web View allows pRTI™ to be installed and deployed in a more robust manor, e.g. in fault server environment.

See Pitch Web View User's guide for how to access it for the first time and how to configure and deploy it.

8 Pitch Control Center

This section describes Pitch Control Center, which is an application running on the logged in desktop of a computer where the Pitch pRTI™ LRC is installed. The purpose is to present useful diagnostics and error messages of LRCs executed by federates on the computer.

8.1 Overview

On most operating systems, the Pitch pRTI™ installer will add a shortcut to Pitch Control Center to the default startup directory of the desktop environment. If this is not the case (typically on Linux desktop environments), Pitch Control Center may be started manually or it may be added manually to the default startup applications list. The executable for Pitch Control Center is to be found in the *bin* subdirectory of the Pitch pRTI™ installation.

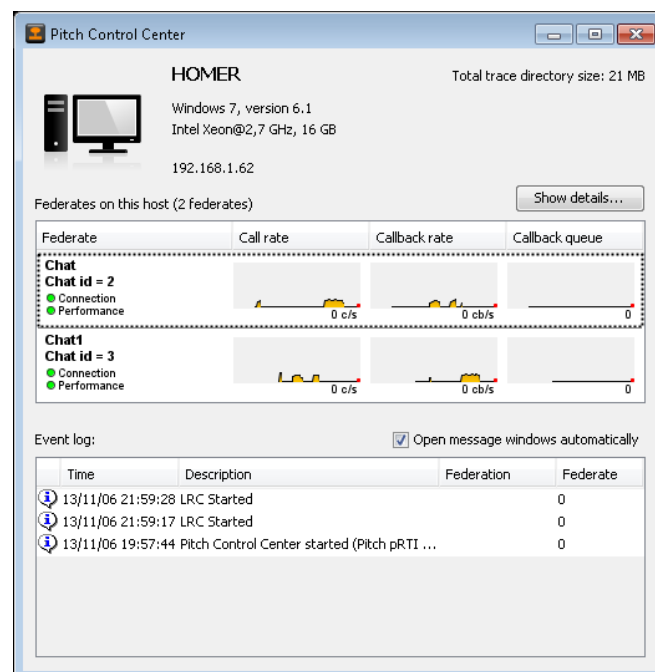


Figure 39 – Pitch Control Center overview

When the Pitch Control Center window is minimized or closed, it will still be running available in the system tray. Right clicking the system tray will provide additional options for starting settings editors, get version information, check for updates or to close Pitch Control Center.

8.2 Friendly Errors

One of the tasks of Pitch Control Center is to present friendly error messages from LRCs of federates running on the computer. So, if an error occurs in a federate connected to Pitch Control Center (which happens automatically if Pitch Control Center is running) the error is presented in a dialog box by Pitch Control Center. The error is also available in the event log of Pitch Control Center and double clicking the line of the event log will reopen the dialog presenting the error message.

The Friendly Errors message presented in the dialog box is not only intended to tell you what went wrong, but also intended to give you some qualified hint of what is a likely cause and solution of such a problem.

8.3 LRC Monitoring

The overview of the Pitch Control Center main window shows some basic information for each federate on the computer:

- A green led to indicate that the LRC has a correct connection to the federation, which is a red led otherwise.
- A performance led which is green when the federate have no obvious performance problems and which turns red in case the federate does not cope with consuming received callbacks fast enough or potentially in the event of other performance problems.
- Rate meters with small graphs displaying the rate of performed calls to the RTI, rate of received callbacks and the size of the call back queue. The callback queue is the queue where incoming callbacks are stored waiting for the federate application to consume them.

All this information and other details can be found by opening the monitoring window of an individual federate as displayed in the next figures.

The status tab shows some general status information.

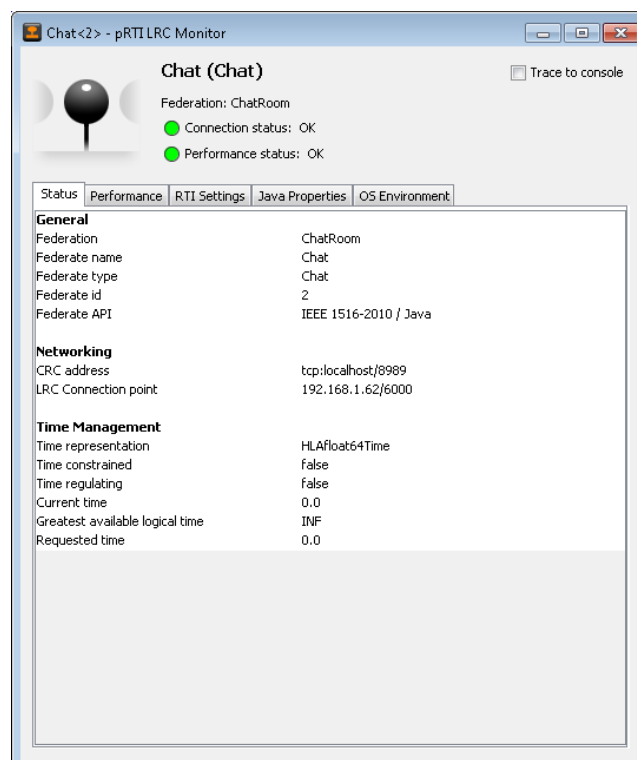


Figure 40 – Pitch Control Center LRC monitor overview

The Performance tab shows performance related information and graphs of the most significant metrics for determining the performance health of a federate. Whenever a value in the table of number turns red, click it to get a hint of what is wrong and why it is considered a performance threatening value.

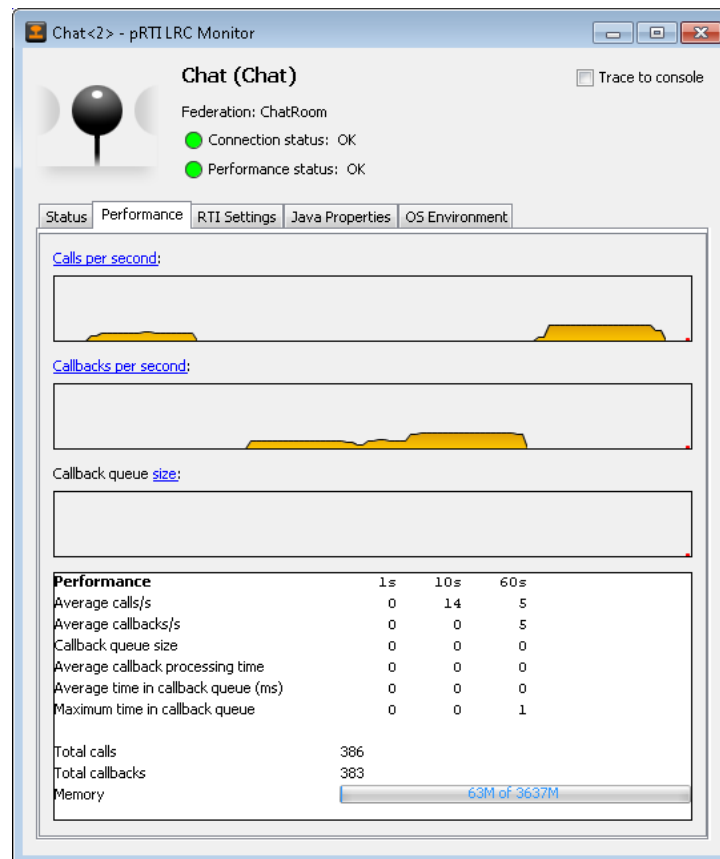


Figure 41 – Pitch Control Center LRC performance monitor

The RTI settings tab shows the LRC settings being used for the LRC. LRC settings may be a combination of different settings files, and this view will help you investigate what the exact value of each setting is and from where it has been loaded.

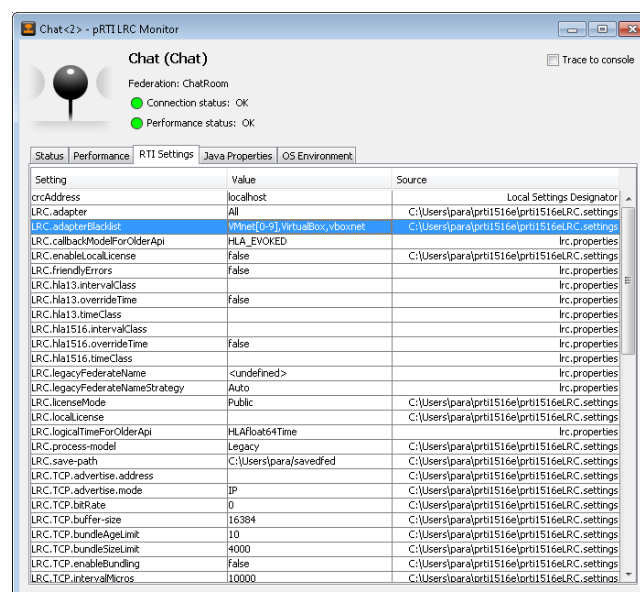


Figure 42 – Pitch Control Center LRC monitor RTI settings

The Java Properties tab shows the Java properties of the LRC. Even though your federate is written in C++, parts of the LRC will be running Java and therefore this

view is presenting all the Java system properties set by the environment for your federate's LRC.

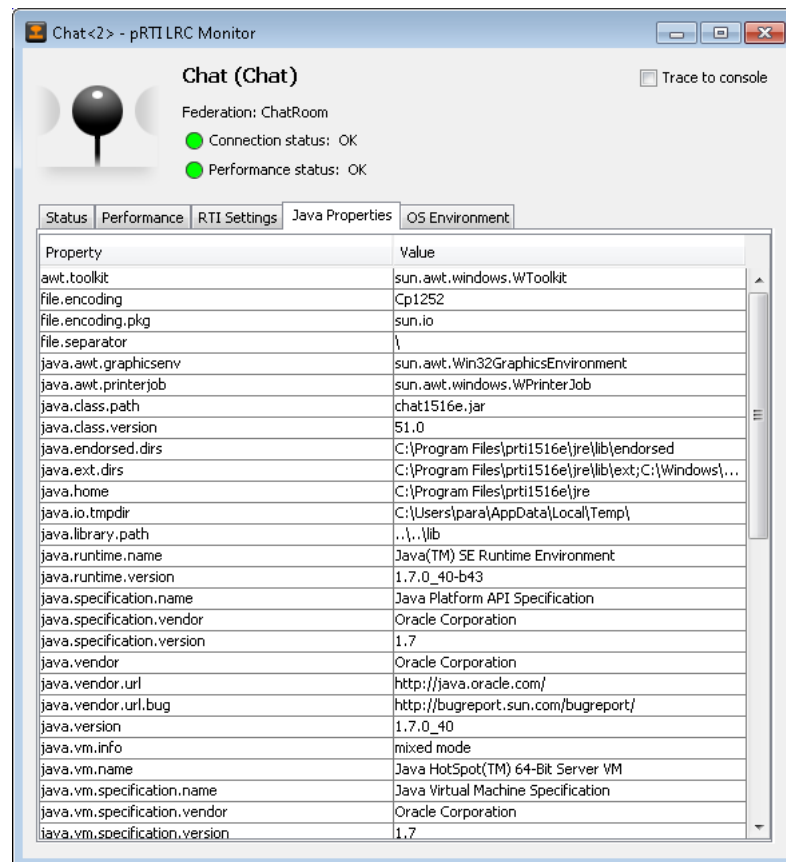


Figure 43 – Pitch Control Center LRC monitor Java properties

The OS Environment tab shows the final result of environment variables in the shell of your federate. Some environment variables most likely being more important than others, such as PATH and CLASSPATH, it still shows the entire list since this is quite individual for each federate application.

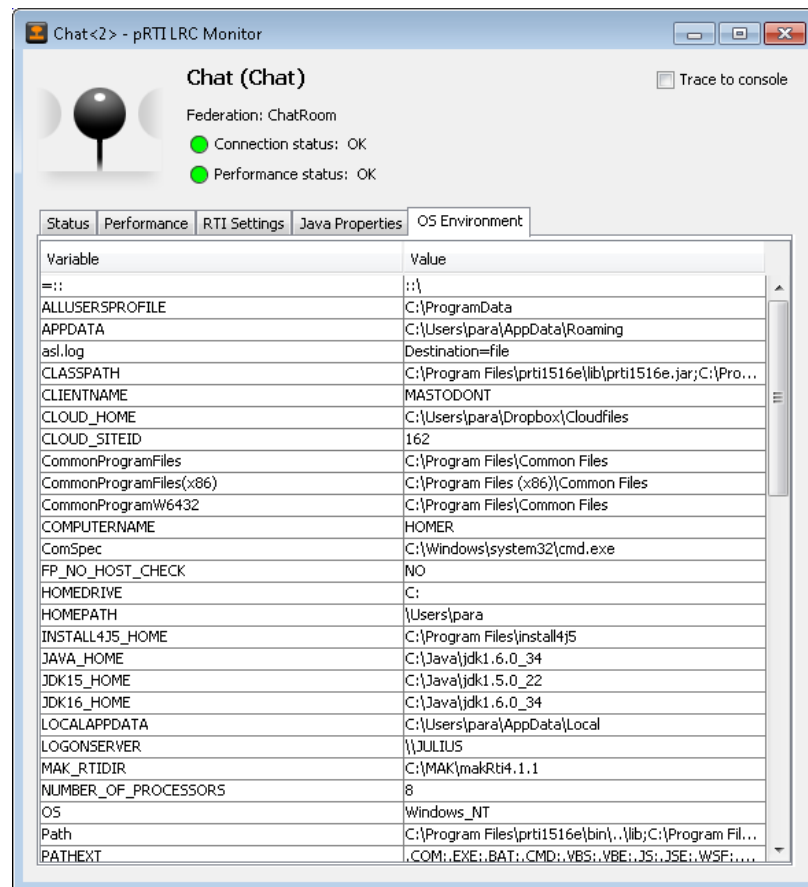


Figure 44 – Pitch Control Center LRC monitor environment variables

9 Developing with Pitch pRTI™

This section describes how you set up your development environment with Pitch pRTI™ so you can start developing federates.

Note that you need to purchase an add-on if you plan to do C++ development. Also note that the DLL:s and shared libraries that are distributed with Pitch pRTI™ are compiled using the compilers mentioned below as well as with other compilers.

This chapter does not cover all compiler versions for which there are pRTI™ libraries. Setting up the development environment is very similar between many of the compatible versions. The *lib* subdirectory of your pRTI™ installation has subdirectories containing libraries for different compiler versions. This is the complete list of supported versions of your installed version of pRTI™. The pRTI™ web site on www.pitch.se also specifies which versions are supported.

If you are using Java, you only need to make sure that you have a Java Development Kit of version 1.6.0 or later.

Technical note: The LRC contains the C++ interface and the Java interface that the federate is using, along with the implementation of some of the functions of Pitch pRTI™. The Windows C++ interface is distributed as a DLL and a .lib file, the Linux C++ interface is distributed as a shared library and the Java interface is distributed as a Java *jar* file.

9.1 Microsoft Visual Studio 2010 on Windows

9.1.1 Creating a New Project

Begin with creating a new project. Under *Visual C++ Projects*, select *Win32 Console Project*. Name the project *chatcc* and place it in the *C:\Program Files\prti1516e\samples* directory. Click OK.

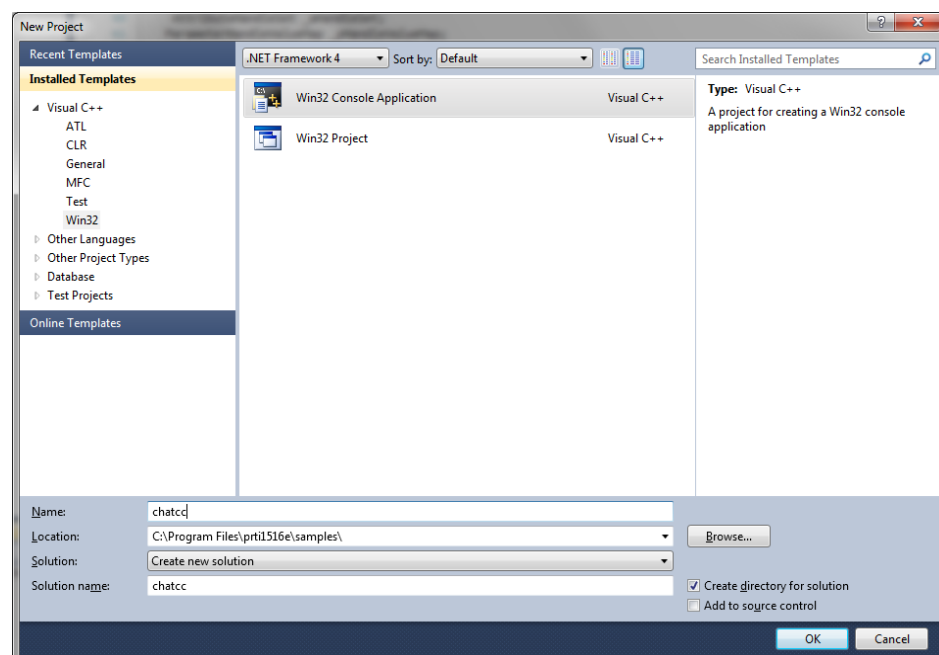


Figure 45 – Creating a new project

Next, select *Application Settings* and make sure that *Console application* and *Empty project* is selected.

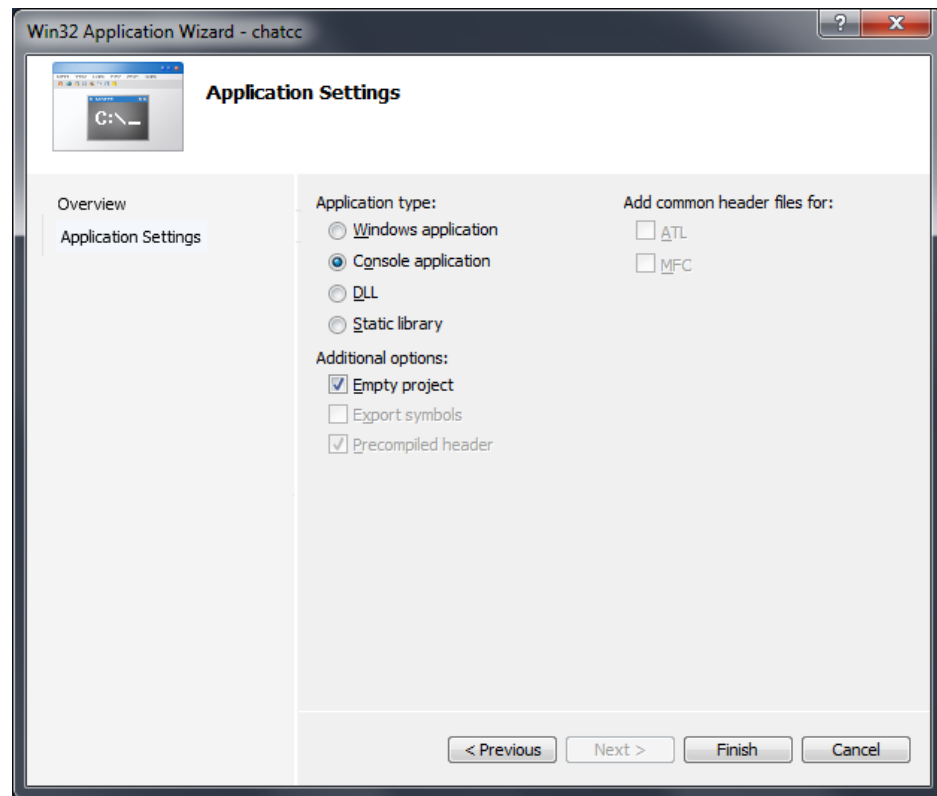


Figure 46 – Application settings for the chatcc project.

From the *Project* menu select *Add Existing Items* and select the files in the *src* directory.

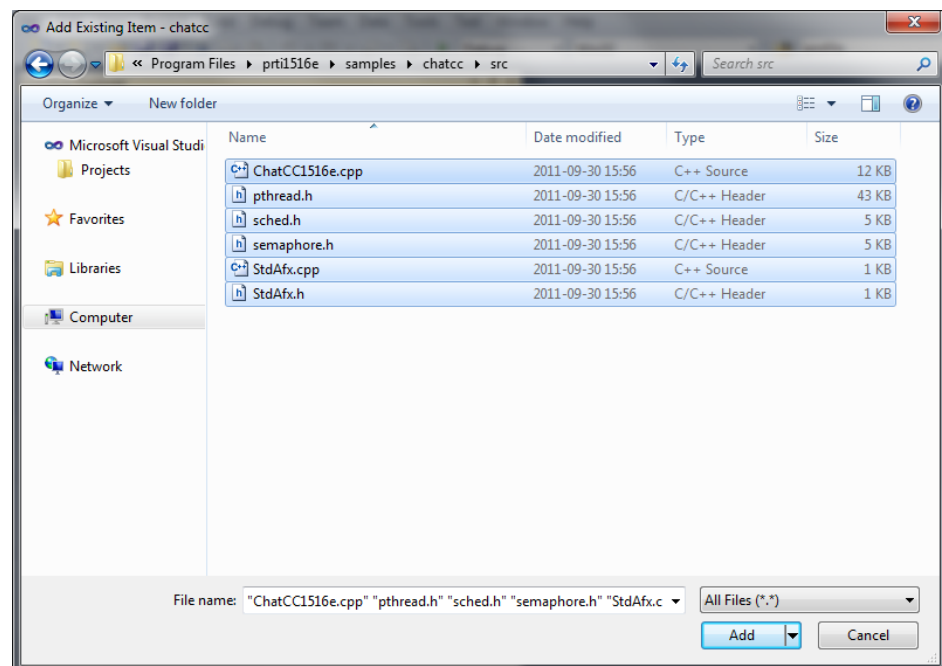


Figure 47 – Add source files to the project.

Now you will need to edit a few project properties. In the *Project* menu select *Properties*. This example will illustrate how to edit the debug configuration

(similar steps are required for the release configuration) so start by selecting *Debug* in the *Configuration* drop-down menu.

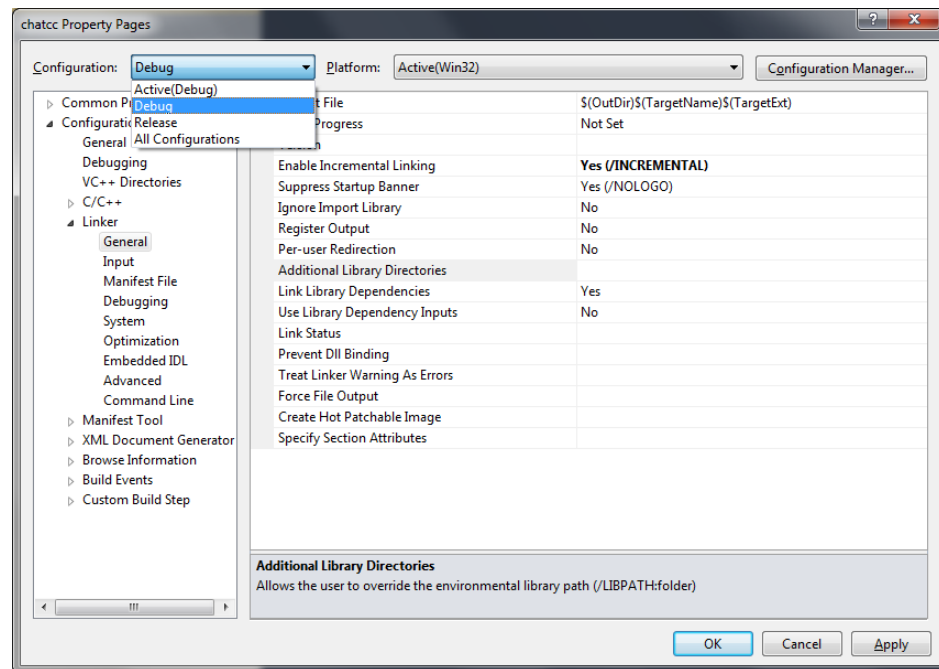


Figure 48 – Settings for the Debug configuration.

9.1.2 Settings Under the C/C++ Folder

In the *Configuration Properties* window click *C/C++* and select *General*. In the *Additional Include Directories* field you will need to specify path to the directory containing the include files for Pitch pRTI™. In the default installation, this is *C:\Program Files\prti1516e\include*. You will also need to include the *src* directory containing the files you previously added: *C:\Program Files\prti1516e\samples\chat-cpp\src*.

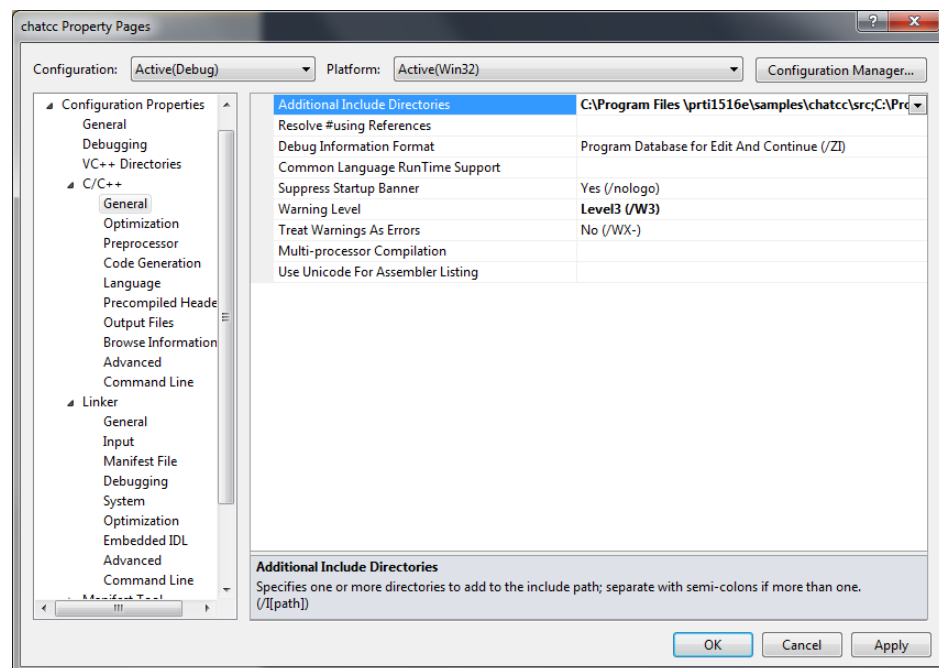


Figure 49 – Specify additional include directories.

Next, select *Code Generation* in the *Configuration Properties* menu. Pitch pRTI™ is a multi-threaded application, so a multi-threaded run-time library is needed. Pitch pRTI™ is distributed with both release and debug versions of the DLL. Since this example illustrates how to edit the debug configuration select *Multi-threaded Debug DLL (/MDd)* in the *Runtime Library* field.

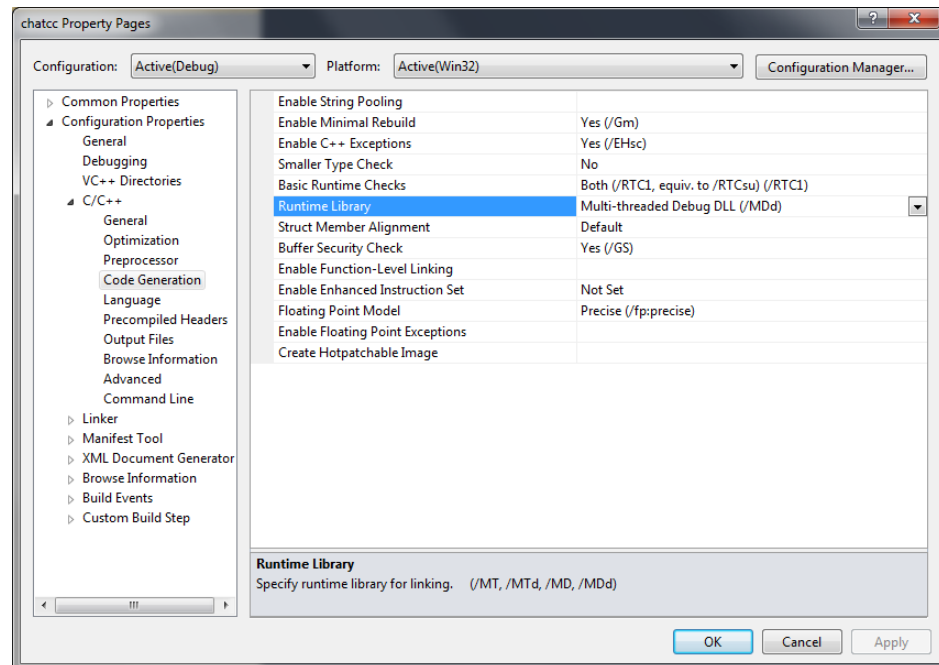


Figure 50 – Select run-time library to use.

Next, select *Language* in the *Configuration Properties* menu and enable *Run-Time Type Info*.

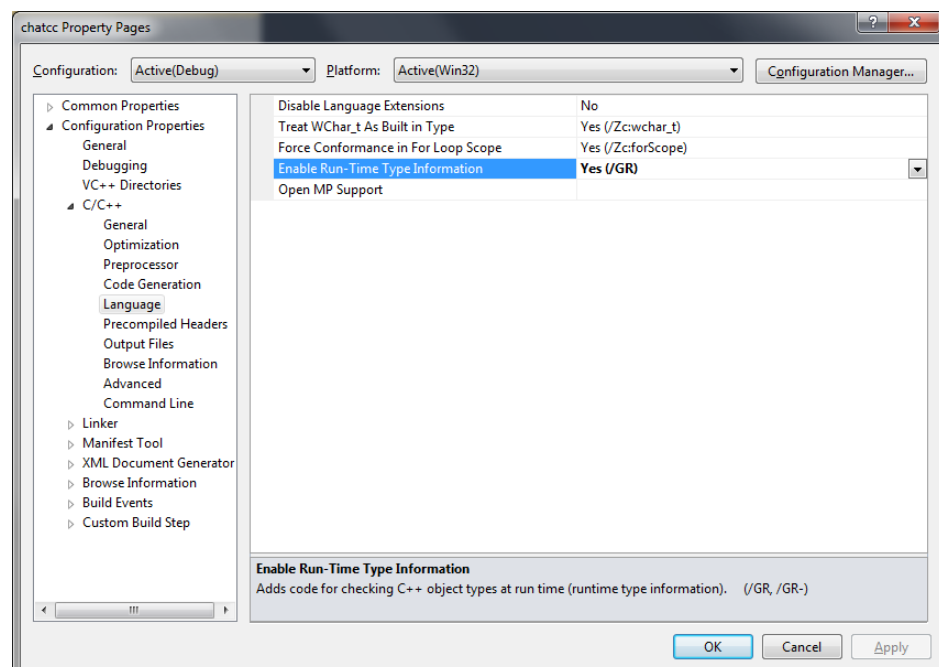


Figure 51 – Enable Run-Time Type Information (RTTI).

These were all the properties that need to be edited under the *C/C++* folder. Next we will need to edit some properties under the *Linker* folder.

9.1.3 Settings Under the Linker Folder

In the *Configuration Properties* window click *Linker* and select *General*. In the *Additional Library Directories* field you will need to specify path to the directory containing the library files for Pitch pRTI™. In the default installation, this is *C:\Program Files\prti1516e\lib\vc100*. You will also need to add the directory containing the *pthread* library: *C:\Program Files\prti1516e\samples\chat-cpp*.

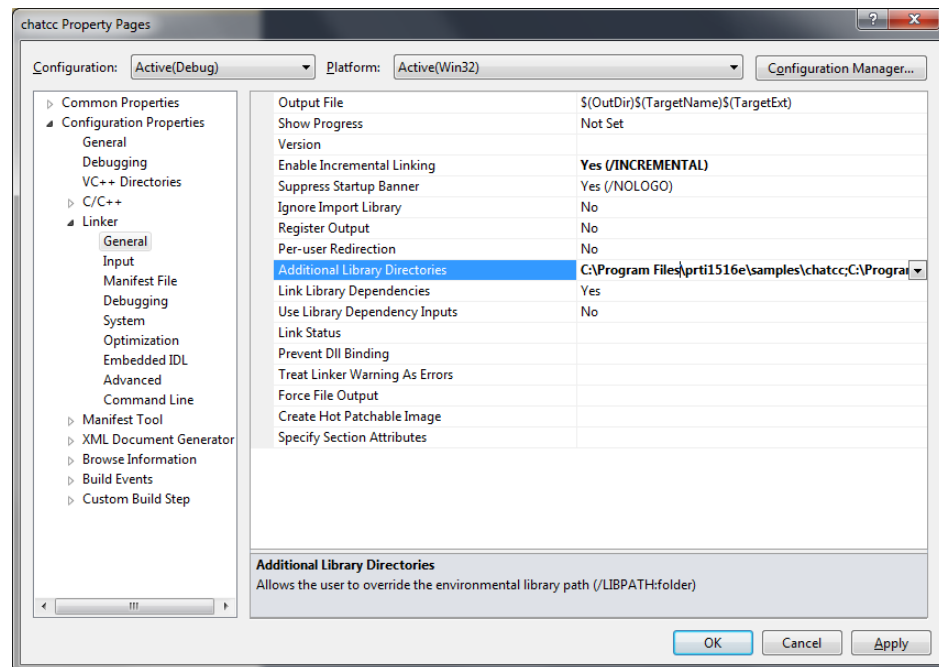


Figure 52 – Specify additional library directories.

Next, select *Input* in the *Configuration Properties* window. In the *Additional Dependencies* field, add the library *librti1516ed.lib*. Note the *d* that appends the *librti1516e* library, this specifies it as a debug library. In the release configuration, *librti1516e.lib* should be specified instead. Also add the library *pthreadVC2.lib* (included with the chatcc sample) which is the threading library used by the sample.

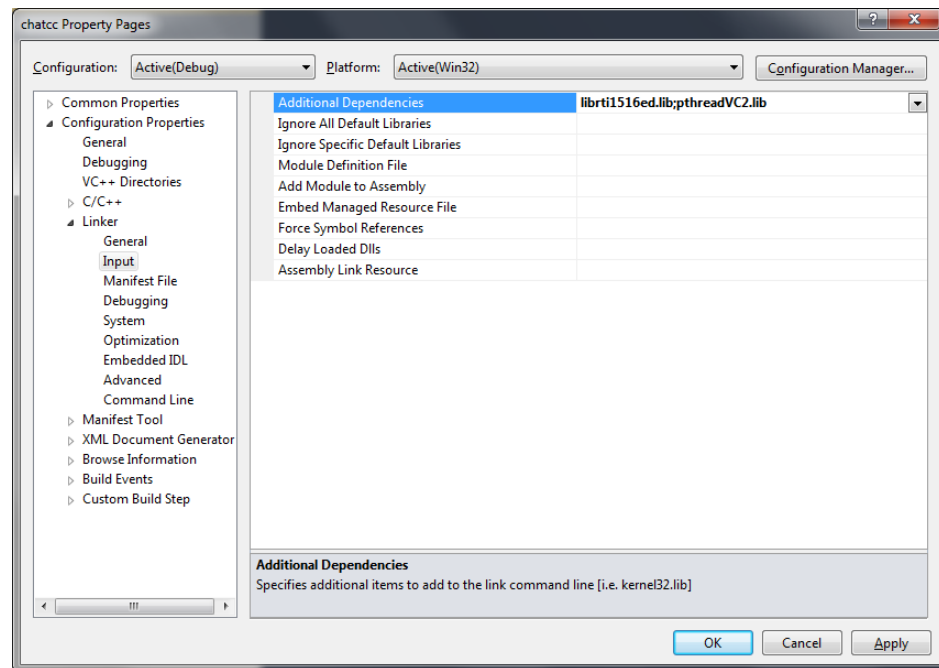


Figure 53 – Specify additional dependencies.

Now you should be able to compile and link the Chat federate.

Do not forget to start Pitch pRTI™ before starting the federate. Also note that the *Chat-evolved.xml* file is assumed to be located in the same directory as the Chat executable.

The setup for the release configuration is similar to that of the debug configuration.

9.2 Optimization flags for Visual Studio release builds

Pitch pRTI™ support the usage of the non-standard compiler flag combination "no_scl" for Microsoft Visual C++ 8.0 and 9.0 compilers on Microsoft Windows OS (Microsoft Visual Studio 2005 and 2008 respectively). The no_scl builds can be used for improved performance or for compatibility with other RTI implementations.

The following, additional, pre-processor flags are set for no_scl builds: "_SECURE_SCL=0" and "_HAS_ITERATOR_DEBUGGING=0". The no_scl builds can be found in the *lib* dir alongside the regular *vc90* and *vc80* libraries, for example *vc90_no_scl*.

9.3 Running federates from the Visual Studio IDE

When executing the federate from the Visual Studio editor you have to configure the debugging environment variables to include the libraries it depends upon by adding them to the PATH. To edit the debugging environment variable in Visual Studio 2010 choose the 'Project' menu item and then 'Properties'. This will bring up the property pages. Under 'Configuration Properties' there's a debugging item where you will find the environment path. Set the path to include all dependencies from the pRTI installation, similar to the linker library path.

9.4 Other Microsoft Visual Studio versions on Windows

The projects configuration steps described for Microsoft Visual Studio 2010 are similar to the configuration of the other supported Visual Studio versions - 6.0, .NET 2003, 2005, and 2008.

9.5 GCC on Linux

This example will illustrate how to set up the C++ chat example provided with the default installation so you can compile it using GCC v4.1.

9.5.1 Modifying the Makefile

In the *src* directory found in the *samples/chat-cpp* directory contains a *Makefile* which can be used to build the ChatCC example federate. Run *make* to build the federate.

9.5.2 Adding prt1516e.jar to the CLASSPATH

To be able to use pRTI™, you need to ensure that the *prt1516e.jar* file is available on your Java CLASSPATH. When using the graphical installer, this is done automatically for you on Windows. The installer adds the *jar* files needed by pRTI™ to the CLASSPATH.

In most cases, we do not recommend moving *prt1516e.jar* from the *lib* subdirectory. But, if that for some reason anyway is necessary it is important to also move the other *jar*-files in the *lib* directory to the same location since there are dependencies between *prt1516e.jar* and those *jar*-files. Note that moving these files will prevent the installer to update your LRC. You will simply have to do that manually.

9.5.3 Modifying the library search path

While looking at the make file, note that a federate is linked with a number of shared libraries. The libraries (including the directory where they can be found) on 32-bit systems are:

- *librti1516e.so* - *\$PRTI_HOME /lib/gcc41*
- *libfedtime1516e.so* - *\$PRTI_HOME/lib/gcc41*
- *libjava.so* - *\$PRTI_HOME /jre/lib/i386/*
- *libverify.so* - *\$PRTI_HOME /jre/lib/i386/*
- *libjvm.so* - *\$PRTI_HOME/jre/lib/i386/client/*

And the similar libraries for 64-bit systems are:

- *librti1516e.so* - *\$PRTI_HOME /lib/gcc41_64*
- *libfedtime1516e.so* - *\$PRTI_HOME/lib/gcc41_64*
- *libjava.so* - *\$PRTI_HOME /jre/lib/amd64/*
- *libverify.so* - *\$PRTI_HOME /jre/lib/amd64/*
- *libjvm.so* - *\$PRTI_HOME/jre/lib/amd64/server/*

When executing the federate, these libraries need to be loaded. The run-time linker needs to be able to find the libraries, and thus it needs to know which directories to search. This can be configured either using the *ldconfig* tool

(available in your Linux environment) or by adding the directories listed above to the environment variable `LD_LIBRARY_PATH`. The easiest thing to do is to add the directories to the `LD_LIBRARY_PATH`.

9.6 Custom signal handlers in C++

If there is a need for custom POSIX signal handlers in the federate application, special care must be taken to ensure correct operation.

Signal handlers are routines that are called when a certain signal is sent. Signal handlers can be registered with the `signal()` system call. Custom signal handlers conflict with signal handlers registered by pRTI and will be disabled.

To make signal handling work correctly the following environment variable must be set:

On windows:

```
set PRTI1516_OPTION1=-Xrs
```

On Linux:

```
export PRTI1516_OPTION1=-Xrs
```

An alternative way of doing this is to create a text file named "prti.vmoptions" in the current working directory of the federate, and add one line with the text "-Xrs" to the file.

9.7 Java

For Java development, all you need to do is to make the file *prti1516e.jar* available in your development environment, and make sure that the file is available on the `CLASSPATH` when your federate is running.

We strongly recommend that you keep the jar file in the pRTI installation and refers the path to it on the `CLASSPATH`. This way upgrades done using an automatic installer keeps your federate up to date and all the other jar-files in the installation on which *prti1516e.jar* may depend on can be resolved. If you however need to move the jar-file to some other location, make sure to also move the other jar files located in the same directory as *prti1516e.jar* (the installation's *lib* directory) to the same location. Upgrading will then require that you manually move these files.

10 Writing a simple federate in C++

This section provides an introduction to writing your own HLA federate in C++. It is not a complete HLA development guideline. A full tutorial for developing HLA federates and federations is available separately on www.pitch.se. It is assumed that you are familiar with the contents of chapter (6) Thus, this chapter does not deal with compiling and linking your federate.

This introduction is based on the source code in the *chatcc* example, provided with the installation of Pitch pRTI™. The *chatcc* example is a small chat application that lets users send messages to each other. The FOM used is shown in Figure 54.

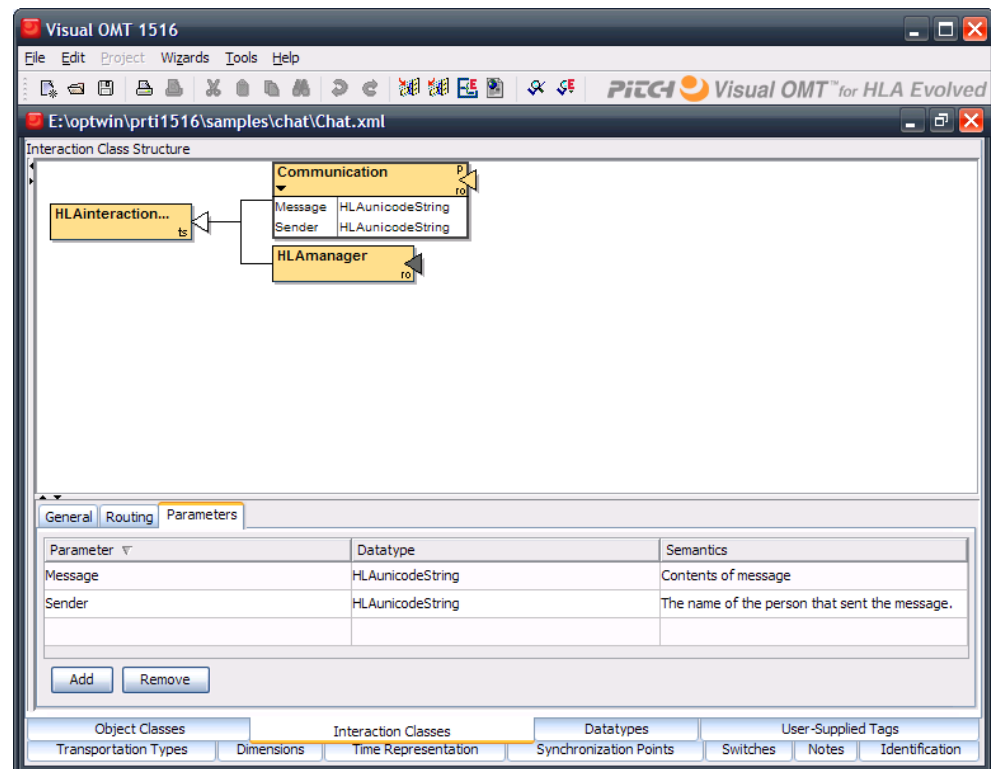


Figure 54 – The FOM used in the ChatRoom federation.

10.1 The FederateAmbassador and the RTIambassador

There are two main classes to deal with when implementing a federate, *FederateAmbassador* and *RTIambassador*.

The *FederateAmbassador* class is the class through which the RTI communicates with the federate. The RTI sends messages to the federate by invoking the methods in the *FederateAmbassador* class (invoking such a method is generally referenced as invoking a callback). When you write your own federate you subclass the abstract *FederateAmbassador* class and implement all its callback methods. The chat samples subclass the *NullFederateAmbassador* class which provides empty implementations for all the callbacks methods in *FederateAmbassador*. When you write your own federate you can simply subclass it and implement (override) the callbacks that you are interested in.

RTIambassador is the class through which the federate communicates with the RTI. It is represented by an auto pointer that is automatically obtained from the *RTIambassadorFactory* class. The code looks like this:

```
auto_ptr< RTIambassador > _rtiAmbassador;
auto_ptr <RTIambassadorFactory> rtiAmbassadorFactory (new RTIambassadorFactory());
_rtiAmbassador = rtiAmbassadorFactory->createRTIambassador();
```

All communication between the federate and the RTI must go through the *FederateAmbassador* and the *RTIambassador* classes.

10.2 Connecting to the RTI

Before creating or joining federation executions, the federate needs to connect to the RTI. This is done through the following call to the *RTIambassador*:

```
wstring settingsDesignator(L"crcAddress="+ host + L":" + port);
_rtiAmbassador->connect(*this, HLA_IMMEDIATE , settingsDesignator);
```

Note that the settings designator string also may be left empty or contain an abstract reference name to a local settings designator.

10.3 The Federation Object Model

Each federation must have one or more *FOM* files (with the extension .xml), which describes the objects and interactions to be used in the federation. The files can easily be created using for example Pitch Visual OMT™.

The first thing that needs to be done is to create the federation execution. The *Create Federation Execution* service takes one or more *FOM* files as argument. This can be done by any federate. If a federation execution with the specified name already exists, an exception is thrown and must be caught. The creation is done with the following code:

```
vector<wstring> FOMmoduleUrls;
FOMmoduleUrls.push_back(L"Chat-evolved.xml");

try {
    _rtiAmbassador->createFederationExecution(L"ChatRoom", FOMmoduleUrls);
} catch (FederationExecutionAlreadyExists federationExecutionAlreadyExists) {
}
```

The parameters are the name of the federation (*ChatRoom*) and a vector of URLs to FOM files.

When the federation is created the federate has to join the federation. This is done with the following code:

```
FederateHandle federateHandle = _rtiAmbassador->joinFederationExecution(
    L"Chat",
    L"ChatType",
    L"ChatRoom",
    FOMmoduleUrls
);
```

The first three parameters are the name of the federate, type of the federate and the federation to join, respectively. The fourth parameter is a vector of URLs to the *FOM* module files used when joining. In our case, adding FOM-modules is unnecessary since we have already created the federation with the same FOM-modules, so this is just a way to illustrate how modules upon join are added.

The federate is now a member of the federation. Note that a federate that creates a federation execution is not automatically joined to that execution.

Objects and interactions are used to exchange data between federates in the federation. Objects have attributes, and interactions have parameters, to describe their characteristics. Only the use of interactions and parameters will be described in this guide.

Each interaction and parameter is represented by a handle. The handles are obtained from the RTI via the *RTIAmbassador* as in the code below.

```
InteractionClassHandle _iMessageId;
ParameterHandle _pTextId;
ParameterHandle _pSenderId;

_iMessageId = _rtiAmbassador->getInteractionClassHandle(L"Communication");
_pTextId = _rtiAmbassador->getParameterHandle(_iMessageId, L"Message");
_pSenderId = _rtiAmbassador->getParameterHandle(_iMessageId, L"Sender");
```

The parameter of the *getInteractionClassHandle* call is the name of the interaction as specified in the *FOM*. The first parameter in the *getParameterHandle* call is the interaction to which the parameter belongs and the second one is the name of the parameter also as specified in the *FOM*.

The handles are the federate's representation of interactions and parameters and can be used when for example sending and receiving interactions.

10.4 Publishing and Subscribing to Information

The exchange of data is controlled by *publishing* and *subscription* of data. For an interaction to be sent, the sending federate must first *publish* it. This means that it tells every federate in the federation execution that it has some information and that it wants to share it. For a federate to receive interactions of a certain class it must *subscribe* to that interaction class. This means that all interactions of the specified classes that are sent will only be delivered to the subscribing federate(s).

You can subscribe to the interaction class *_iMessageId* using the following code:

```
_rtiAmbassador->subscribeInteractionClass(_iMessageId);
```

You will now receive all interactions of class *_iMessageId* that are published and sent by other federates.

To publish the same interaction class, you use the following code:

```
_rtiAmbassador->publishInteractionClass(_iMessageId);
```

Other federates will now receive interactions sent by you (if they have subscribed to them).

10.5 Sending Interactions

Before you send your interactions you must create a *ParameterHandleValueMap* and add the interaction's parameters and the values of them encoded according to the *FOM*. The code looks like this:

```
ParameterHandleValueMap parameters;
HLAUnicodeString hlaMessage(wstr_message);
HLAUnicodeString hlaSender(wstr_sender);
parameters[_pTextId] = hlaMessage.encode();
parameters[_pSenderId] = hlaSender.encode();
```

wstr_message and *wstr_sender* are the values of the parameters you want to send (must be a *wstring*).

To send the interaction the following call is made:

```
_rtiAmbassador->sendInteraction(_iMessageId, _pHandleValueMap, VariableLengthData());
```

The first parameter is the interaction class handle, the second is the *ParameterHandleValueMap*, holding the parameter values of the interaction and the third is a user-supplied tag, in this case not set.

10.6 Receiving Interactions

When the RTI wants to send data to the federate it uses the methods (called callbacks) specified in the *FederateAmbassador* class. In the *chatcc* example there is only one callback implemented, namely *receiveInteraction*, which is called by the RTI when an interaction sent by another federate is to be delivered to your federate.

The *receiveInteraction* callback looks like this:

```
void receiveInteraction (
    InteractionClassHandle theInteraction,
    ParameterHandleValueMap const & theParameterValues,
    VariableLengthData const & theUserSuppliedTag,
    OrderType sentOrder,
    TransportationType theType,
    SupplementalReceiveInfo theReceiveInfo)
    throw (FederateInternalError)
```

The first parameter is the class of the received interaction and the second one is the interaction's parameters. The last three are not of interest here.

To get the parameter values out of the *ParameterHandleValueMap* you can for example go through the map using a loop like this:

```
for (ParameterHandleValueMap::const_iterator i = theParameterValues.begin();
     i != theParameterValues.end(); ++i) {
    ParameterHandle const & handle = i->first;
    VariableLengthData const & value = i->second;
    if (handle == _pTextId) {
        message = HLAUnicodeString(value);
    } else if (handle == _pSenderId) {
        sender = HLAUnicodeString(value);
    }
}
```

For each entry in the map you check the parameter handle. In this example, the parameter value is decoded as a *HLAUnicodeString* according to the *FOM*.

10.7 Conclusions

It is very easy to create a small federate. However, if you want to learn to use the full potential of HLA, you should attend a *HLA Hands On* development course. More information about this can be found at <http://www.pitch.se>.

11 Writing a Simple Federate in Java

This section provides an introduction to writing your own HLA federate in Java. It is not a complete HLA development guideline. A full tutorial for developing HLA federates and federations is available separately on www.pitch.se. It is assumed that you are familiar with the contents of chapter (6) Thus, this chapter does not deal with compiling and linking your federate.

This introduction is based on the source code in the *chat* example provided with the installation of Pitch pRTI™. The *chat* example is a small chat application that lets users send messages to each other. The FOM used is shown in Figure 55.

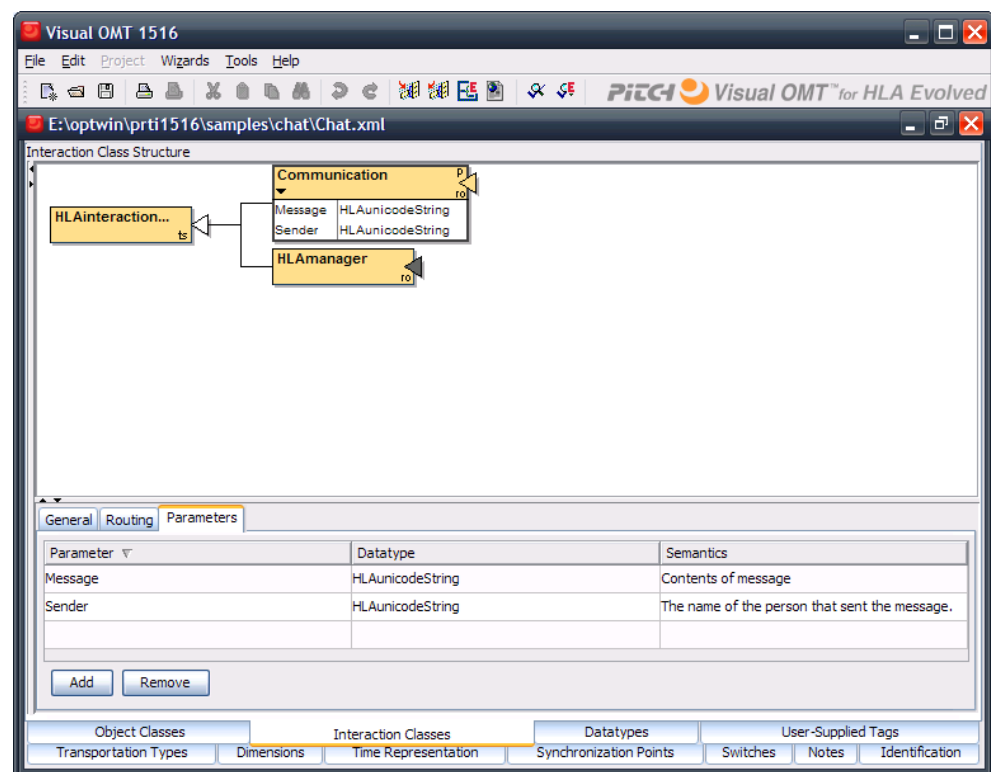


Figure 55 – The FOM used in the ChatRoom federation.

11.1 The FederateAmbassador and the RTIambassador

There are two main Java interfaces to deal with developing a federate, *FederateAmbassador* and *RTIambassador*.

The *FederateAmbassador* interface is the interface through which the RTI communicates with the federate. The RTI sends messages to the federate by invoking the methods in the *FederateAmbassador* class (invoking such a method is generally referenced as invoking a callback).

When you write your own federate you implement the *FederateAmbassador* interface with a class that implements all the available callback methods. There is also the convenience class *NullFederateAmbassador* which simply provides empty implementations of all the callback methods in the *FederateAmbassador* interface. When you develop your own federate you can simply subclass *NullFederateAmbassador* and implement (override) the callbacks that you are interested in.

RTIambassador is the class through which the federate communicates with the RTI. The code looks like this:

```
RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory();
_rtiAmbassador = rtiFactory.getRtiAmbassador();
```

The first line is the default way of retrieving an *RtiFactory* (multiple RTI-factories may be available depending on which RTI:s that is installed on the computer). The second line calls the *RtiFactory* to get an *RTIambassador*.

All communication between the federate and the RTI must go through the *FederateAmbassador* and the *RTIambassador* interfaces.

11.2 Connecting to the RTI

Before creating and joining federation executions, the federate must connect to the RTI. This is done through the following code:

```
String settingsDesignator = "crcAddress=" + rtiHost + ":" + Integer.toString(CRC_PORT);
_rtiAmbassador.connect(this, CallbackModel.HLA_IMMEDIATE, settingsDesignator);
```

Note that the settings designator string also may be left empty or contain an abstract reference name to a local settings designator.

11.3 The Federation Object Model

Each federation must have one or more *FOM* files (with the extension .xml), which describe the objects and interactions to be used in the federation. The files can easily be created using for example Visual OMT™ 1516.

The first thing that needs to be done is to create the federation execution. The *Create Federation Execution* service takes one or more *FOM* files as an argument. This can be done by any federate. If a federation execution with the specified name already exists, an exception is thrown and must be caught. The creation is done with the following code:

```
File fddFile = new File("Chat-evolved.xml");
try {
    _rtiAmbassador.createFederationExecution("ChatRoom", fddFile.toURL());
} catch (FederationExecutionAlreadyExists ignored) {
}
```

The parameters are the name of the federation (*ChatRoom*) and the URL representation of the single *FOM* file to use in this case (*Chat-evolved.xml*).

When the federation is created the federate has to join the federation. This is done with the following code:

```
FederateHandle federateHandle = _rtiAmbassador.joinFederationExecution("Chat", "ChatType",
    "ChatRoom", new URL[]{fddFile.toURL()});
```

The first three parameters are the name of the federate, type of the federate and the name of the federation to join, respectively. The fourth parameter is an array of URLs to *FOM* files that the federate wants to bring to the federation. In our case, adding *FOM*-modules is unnecessary since we have already created the federation with the same *FOM*-modules, so this is just a way to illustrate how modules upon join are added.

The federate is now a member of the federation. Note that a federate that creates a federation execution is not automatically joined to that execution.

Objects and interactions are used to exchange data between federates in the federation. Objects have attributes, and interactions have parameters, to describe their characteristics. Only the use of interactions and parameters will be described in this guide.

Each interaction and parameter is represented by a handle. The handles are obtained from the RTI via the *RTIAmbassador* as in the code below.

```
InteractionClassHandle _messageId;
ParameterHandle _parameterIdText;
ParameterHandle _parameterIdSender;

_messageId = _rtiAmbassador.getInteractionClassHandle("Communication");
_parameterIdText = _rtiAmbassador.getParameterHandle(_messageId, "Message");
_parameterIdSender = _rtiAmbassador.getParameterHandle(_messageId, "Sender");
```

The parameter of the *getInteractionClassHandle* call is the name of the interaction as specified in the *FOM* file. The first parameter in the *getParameterHandle* call is the interaction to which the parameter belongs and the second one is the name of the parameter.

The handles are the federate's representation of interactions and parameters, and can be used when for example sending and receiving interactions.

11.4 Publishing and Subscribing to Information

The exchange of data is controlled by *publishing* and *subscription* of data. For an interaction to be sent the sending federate must first *publish* it, which means that it tells everyone that it has some information and that it wants to share it. For a federate to receive interactions of a certain class it must *subscribe* to that interaction class. This means that all interactions of the specified classes that are sent will only be delivered to the subscribing federate(s).

You can subscribe to the interaction class *_messageId* using the following code:

```
_rtiAmbassador.subscribeInteractionClass(_messageId);
```

You will now receive all interactions of class *_messageId* that are published and sent by other federates.

To publish the same interaction class, you use the following code:

```
_rtiAmbassador.publishInteractionClass(_messageId);
```

Other federates will now receive interactions sent by you (if they have subscribed to them).

11.5 Sending Interactions

Before you send your interactions you must create a *ParameterHandleValueMap* and add the interaction's parameters and the values of them encoded according to the *FOM*. The code looks like this:

```
ParameterHandleValueMap parameters;
parameters = _rtiAmbassador.getParameterHandleValueMapFactory().create(1);
HLAUnicodeString messageEncoder = _encoderFactory.createHLAUnicodeString();
HLAUnicodeString nameEncoder = _encoderFactory.createHLAUnicodeString();
messageEncoder.setValue(_message);
nameEncoder.setValue(_username);
parameters.put(_parameterIdText, messageEncoder.toByteArray());
parameters.put(_parameterIdSender, nameEncoder.toByteArray());
```

`_message` and `_username` are the values of the parameters you want to send. The *ParameterHandleValueMap* is based on the *java.util.Map* interface and uses a *ParameterHandle* as the key and a value as the value.

To send the interaction the following call is made:

```
_rtiAmbassador.sendInteraction(_messageId, parameters, null);
```

The first parameter is the interaction class handle, the second is the *ParameterHandleValueMap*, holding the parameter values of the interaction and the third is a user-supplied tag, in this case set to null.

11.6 Receiving Interactions

When the RTI wants to send data to the federate it uses the methods (called callbacks) specified in the *FederateAmbassador* class. The methods are all empty in the *FederateAmbassadorImpl* class, which means that you only have to implement the callbacks you are interested in your subclass. In the chat example there is only one callback implemented, namely *receiveInteraction*, which is called by the RTI when an interaction sent by another federate is to be delivered to your federate.

The *receiveInteraction* callback looks like this:

```
public void receiveInteraction(InteractionClassHandle interactionClass,
                             ParameterHandleValueMap theParameters,
                             byte[] userSuppliedTag,
                             OrderType sentOrdering,
                             TransportationTypeHandle theTransport,
                             SupplementalReceiveInfo receiveInfo)
```

The first parameter is the class of the received interaction and the second one is the interaction's parameters. The last four are not of interest here.

To get the parameter values out of the *ParameterHandleValueMap* you can for example go through the map using a for loop like this:

```
String message;
String sender;
for (Iterator i = theParameters.keySet().iterator(); i.hasNext(); ) {
    ParameterHandle parameterHandle = (ParameterHandle) i.next();
    if (parameterHandle.equals(_parameterIdText)) {
        HLAUnicodeString messageDecoder = _encoderFactory.createHLAUnicodeString();
        messageDecoder.decode((byte[]) theParameters.get(_parameterIdText));
        message = messageDecoder.getValue();
    }
    else if (parameterHandle.equals(_parameterIdSender)) {
        HLAUnicodeString senderDecoder = _encoderFactory.createHLAUnicodeString();
        senderDecoder.decode((byte[]) theParameters.get(_parameterIdSender));
        sender = senderDecoder.getValue();
    }
}
System.out.println(sender + ": " + message);
```

For each entry in the map you check the parameter handle. In this example, the parameter value is decoded according to the *FOM*.

11.7 Conclusions

It is very easy to create a small federate. However, if you want to learn to use the full potential of HLA, you should attend a *HLA Hands On* development course.

More information about this can be found at <http://www.pitch.se>.

12 Tick and Process Models

Whether you are migrating from another RTI or developing new federates you will need to think about the process model. This will affect the federate developer in the following ways:

- It will affect the performance and responsiveness of the federation as well as how difficult it will be to tune the final federation.
- It will affect the way the program code is structured.

12.1 Setting the process model

When connecting to the RTI using the RTI-ambassador call *connect* you may have noticed that one of the parameters is an callback-model enumerator. This parameter sets the federate's process model. HLA Evolved has two types of process models:

- **HLA_IMMEDIATE**, which provides a multi threaded process model.
- **HLA_EVOKED**, which is a single threaded, or "evoked", process model.

Using the evoked process model, callbacks are not immediately delivered, but instead only delivered when the federate makes the RTI-ambassador calls *evokeMultipleCallbacks()* or *evokeCallback()*.

12.2 Practical Guidelines

If you are migrating a federate from an RTI which is not multi-threaded such as RTI:NG you have two choices:

- **For the fastest porting:** Disable multi-threading by simply setting the callback model to *HLA_EVOKED*. Tune parameters and tick frequency until you get satisfactory performance.
- **For the best performance and responsiveness:** Set the callback model to *HLA_IMMEDIATE*. Make sure that your federate can handle callbacks anytime. Remove or disable calls to tick since they are unnecessary and may consume CPU resources.

If you are developing a new federate:

- Simply design your federate so that it can handle callbacks anytime. Use *HLA_IMMEDIATE*. Do not call tick or make a call that can easily be disabled.

You are also recommended to limit the amount of work that your federate does in the callback from the RTI. A federate which spends a lot of time in the callback may reduce the performance for operations that are coordinated across the federation, such as time management.

12.3 Explanation of Process Models

The process model can be explained in the following way: There is a Local RTI Component running on the same computer and usually in the same process as your federate (simulation). It needs to do internal work, such as reading incoming network information. It also needs to deliver information to your simulation. So when can the RTI perform its internal processing?

- In a *single-threaded* RTI it will only perform internal processing when you call the tick function. Callbacks to your federate will also be delivered during this call. If you do not call the tick function with the optimal frequency and optimal parameters the performance will suffer. These parameters will vary depending on federates, hardware, network, scenario and more. The RTI may also suffer from starvation if tick is not called often enough. Few RTI implementations rely on this method today.
- In an *asynchronous* RTI the internal processing will be done automatically, independent of your tick calls. Callbacks will only be delivered when you call tick. The performance tuning issue remains but there is no risk for starvation.
- In a *multithreaded* RTI the internal processing will be done automatically. Callbacks will be delivered as soon as new information is available. There is no need to call tick and no tick frequency or parameters to tune. The performance and responsiveness will be optimal from the RTI standpoint but will still of course be dependent on your federate.

The multithreaded strategy is recommended but the asynchronous strategy is also supported by Pitch pRTI™. The tick strategy only affects the Local RTI Component. A federation using Pitch pRTI™ may mix federates with different process models. A federate which is not multithreaded and with inferior tick tuning, may reduce the performance of the entire federation.

Read more about process models in the *SIW* paper *03S-SIW-055* available from *SISO* (<http://www.sisostds.org>).

13 Debugging and Tracing

13.1 Overview

The call tracing feature allows you to inspect the communication between a federate and the Local RTI Component. This communication may occur according to the following pattern:

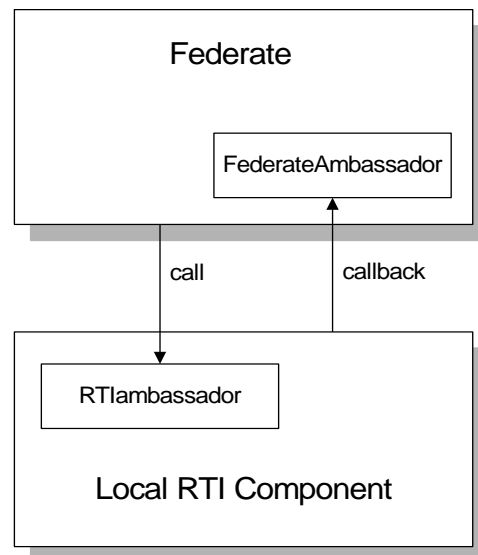


Figure 56 – Calls and callbacks.

The federate initially calls the *RTI Ambassador* to create a federation execution and to join it. It then calls the *RTI Ambassador* to declare its need to produce and consume information (publish and subscribe). During the execution it then calls the *RTI Ambassador* to register new objects, send updates for attributes, send interactions, etc. If you are a federate developer your program will call the methods of the *RTI Ambassador*.

The RTI delivers callbacks to the *Federate Ambassador*, for example information about new objects (discoveries) and updates and interactions received from other federates. If you are a federate developer you will need to implement a *Federate Ambassador* to handle the information that will be delivered to your federate.

Calls and callbacks can be logged using the call-tracing feature. This is useful for example:

- When you develop your federate to make sure that your program interacts with the RTI in the way that you intended.
- When you test your entire federation to analyze for example which information your federate sends and what information was delivered to your federate.

There is no functionality to store large amounts of data into databases or playing back log files from previous sessions. There are however other tools on the market that can do this for you.

13.2 Enabling the Tracing

The recommended way to configure trace settings is to use the *Trace Settings* graphical editor which can be found in the start menu or in the *bin* directory in your Pitch pRTI™ installation.

The trace settings editor opens a file called *pRTI1516e.logging* in the *pRTI1516e* subdirectory of your home directory (e.g. *C:\Documents and Settings\username\pRTI1516e* on Windows). Editing this file using the trace settings editor will enable you to trace the federation startup.

Specify a filename (including the absolute path, e.g. *C:\mylog.txt*) instead of *<console>* to make the trace output appear in a file. Note the extra backslash (\) characters that are required when specifying a file.

13.3 Format of the Trace Log

The format of the trace log is as follows:

<Timestamp>: **<Federate number>** **<Direction>** **<Method>**(**<Parameters>**) => **<Return value>**

Timestamp is given in milliseconds for the local computer. It is not guaranteed to be synchronized between different computers.

Federate number is the number that the federate was given when it joined the federation. Initially it will be zero (before joining).

Direction is << for calls from the Federate to the RTI Ambassador and >> for calls from the RTI to the Federate Ambassador.

Method is the name of the method, for example *joinFederationExecution*.

Parameters is the list of parameter values. In case this is your *FOM* data the hex value will be displayed since the RTI has no knowledge about the correct interpretation.

Return value may also be shown in some cases.

13.4 A Sample Trace Log

The sample trace log below shows parts of the log of one of the chat federates in the installation verification from section 3.3 in this manual.

Notice the *receiveInteraction* with the hex code (marked with boldface) for “Hello Fred” in ASCII. Hint: space=20, A=40, a=60.

```
05:53:08:498 (2005.03.03): fed0 << destroyFederationExecution(ChatRoom) => Federation
Execution Does Not Exist (909146015)
05:53:09:107 (2005.03.03): fed0 << createFederationExecution(ChatRoom, file:/C:/Program
Files/pRTI1516/samples/chat/Chat.xml)
05:53:09:811 (2005.03.03): fed2 << joinFederationExecution(Chat, ChatRoom,
se.pitch.Chat1516.Chat@13adc56, null) => federate<2>
05:53:09:842 (2005.03.03): fed2 << getInteractionClassHandle(Communication) => Interaction
class<2>
05:53:09:842 (2005.03.03): fed2 << getParameterHandle(Interaction class<2>, Message) =>
parameter<100>
05:53:09:842 (2005.03.03): fed2 << getParameterHandle(Interaction class<2>, Sender) =>
parameter<101>
05:53:09:842 (2005.03.03): fed2 << subscribeInteractionClass(Interaction class<2>)
05:53:09:857 (2005.03.03): fed2 << publishInteractionClass(Interaction class<2>)
05:53:09:857 (2005.03.03): fed2 << getObjectClassHandle(Participant) => objectClass<5>
05:53:09:857 (2005.03.03): fed2 << getAttributeHandle(objectClass<5>, Name) =>
attribute<139>
```

```
05:53:09:857 (2005.03.03): fed2 << subscribeObjectClassAttributes(objectClass<5>,
{attribute<139>})
05:53:09:857 (2005.03.03): fed2 << publishObjectClassAttributes(objectClass<5>,
{attribute<139>})
05:53:13:514 (2005.03.03): fed2 << reserveObjectInstanceName(Fred)
05:53:13:529 (2005.03.03): fed2 >> objectInstanceNameReservationSucceeded(Fred)
05:53:13:529 (2005.03.03): fed2 << registerObjectInstance(objectClass<5>, Fred) =>
instance<101>
05:53:13:545 (2005.03.03): fed2 << updateAttributeValues(instance<101>, {attribute<139>,
[46726564]}, [])
05:53:13:561 (2005.03.03): fed2 << requestAttributeValueUpdate(objectClass<5>,
{attribute<139>}, [])
05:53:39:310 (2005.03.03): fed2 >> discoverObjectInstance(instance<103>, objectClass<5>,
Barney)
05:53:39:342 (2005.03.03): fed2 >> provideAttributeValueUpdate(instance<101>,
{attribute<139>}, [])
05:53:39:342 (2005.03.03): fed2 >> reflectAttributeValues(instance<103>, {attribute<139>,
[4261726e 6579]}, [], OrderType(1), TransportationType(1))
05:53:39:357 (2005.03.03): fed2 << updateAttributeValues(instance<101>, {attribute<139>,
[46726564 00]}, [])
05:53:41:920 (2005.03.03): fed2 >> receiveInteraction(Interaction class<2>, {parameter<100>,
[48656c6c 6f204672 656400], parameter<101>, [4261726e 657900]}, [], OrderType(1),
TransportationType(1))
```

14 Networking

This section covers advanced networking topics. For most Pitch pRTI™ users there will seldom be any need to read this chapter. The default networking settings for Pitch pRTI™ will yield the best performance and compatibility for many needs.

Before you get into network tuning you should consider the following:

1. Be sure that you have designed the federation in the spirit of HLA. Do not try to replicate all of the information between all federates, only subscribe to the necessary objects, attributes and interactions. Use DDM where appropriate to limit the information further.
2. The biggest performance gain is usually achieved by simply adding faster networking hardware for example by upgrading from 10 Mbit networking to 100 Mbit or from 100 Mbit to Gigabit Ethernet.
3. One of the big advantages with Pitch pRTI™ compared to many other RTI:s is that it uses sender side filtering. Only the necessary information is sent to federates that subscribe to it. This means that by simply replacing hubs with switches you may experience big performance improvements.
4. To achieve higher responsiveness and throughput in your federation you are also highly recommended to use multi-threaded federates instead of ticked federates. Note that a federation may mix the two types of federates. Read more in chapter 12.

14.1 When to Reconfigure Networking

There are a few situations where advanced users may want to change the default settings:

1. You have several network interface cards and you do not want to run RTI communications on all of them.
2. You need to configure Pitch pRTI™ to run through firewalls. Note that since firewalls are intended to stop unwanted communications you will need to do this in cooperation with your network administrator.
3. You have extremely high requirements for throughput or update rate and you are willing to sacrifice some compatibility with wide area networks.
4. You are losing too many best effort messages.

Reconfiguring Pitch pRTI™ networking will not have any impact on situations where:

1. The latency (delay) on the network between two sites is too large.
2. The total amount of information that a federate subscribes to exceed the capacity of the link to that federate.
3. Some federates can only process a limited amount of incoming information per second, thus lowering the throughput or update rate. It may also be the case that some federates do a lot of work in the RTI callbacks, thus reducing the number of updates and other RTI operations per second (federate callback latency).

The rest of this chapter assumes that you are familiar with networking concepts such as TCP/IP and UDP/IP protocols, port numbers, unicast and multicast, routing and firewalls.

14.2 Overview of Pitch pRTI™ Communication

When the pRTI™ components (CRC and LRC:s) of a federation start, they will establish communication links with each other. The following types of communication links are used:

1. RTI components in the same process automatically use shared memory queues for reliable communication. One such example is LRC:s of two federates executing in the same Java Virtual Machine.
2. RTI components in different processes and possibly on different computers use TCP/IP for reliable communication point-to-point.
3. UDP/IP (point-to-point or multicast) is used for best effort.

All the communication links for a federation can be inspected in the Pitch pRTI™ GUI in the *Network Info* pane as Figure 57 shows.

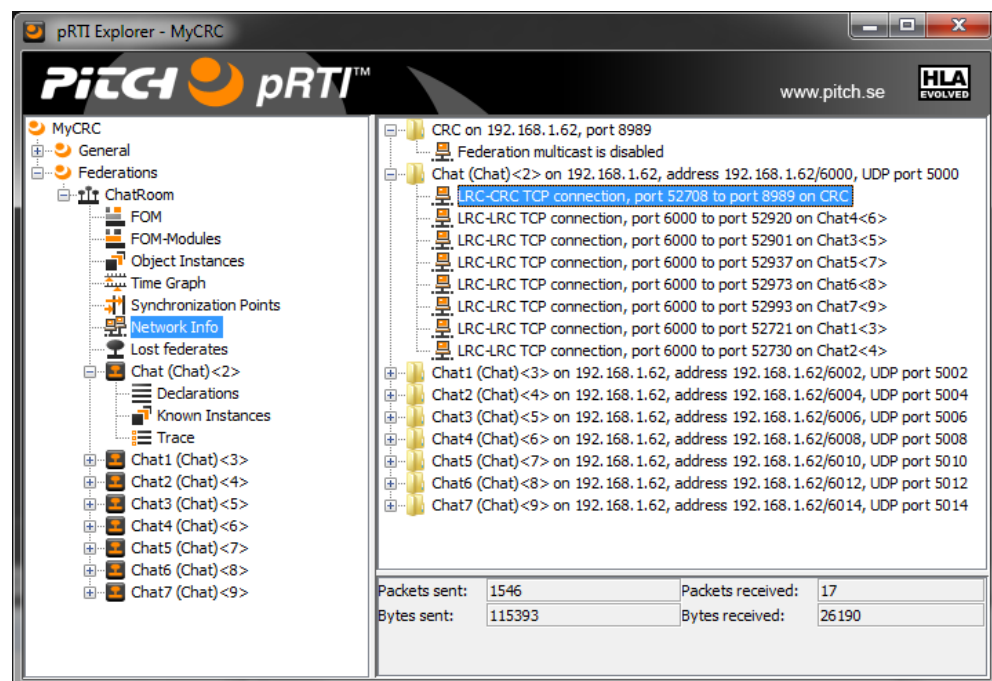


Figure 57 – The network connections between the LRC:s and the CRC.

You can see the following network links:

- *CRC-LRC TCP Connection*, which is a reliable connection between the CRC and LRC. One such connection will be made for each LRC. For each such connection there is also a corresponding *LRC-CRC TCP Connection*.
- *LRC-LRC TCP Connection*, which is a reliable connection between two federates. Each LRC will make one connection of this type to each one of the other LRC:s.

Note that for the CRC there is a *Master TCP port* that the CRC listens on, by default 8989. There may also be a *Multicast address and port* if multicast is enabled.

For the LRC:s there are two *Master ports*, one for TCP (by default 6000) and one for UDP (by default 5000).

The CRC network address and master port as well as the multicast address and port may be configured in the GUI. See section 16.1 for more information.

The LRC network address and ports may be configured for each LRC. The default port range for TCP is 6000 – 6999 and for UDP 5000 – 5999. An LRC will usually use several TCP ports but only one UDP port. See section 16.2 for more information.

14.3 Using Multicast

You can switch on multicast for best effort communication. This means that the information is sent once in a packet that many federates can receive instead of several transmissions, one for each federate.

Advantages:

- Efficient when many federates subscribe to the same information since it is only sent once.

Disadvantages:

- If there are routers between federates, they will probably need to be reconfigured to let multicast traffic through.
- You may actually increase the network load as well as the workload for each LRC since federates may now receive information that they do not need.

When multicast is enabled, Pitch pRTI™ will automatically sense when many federates start to subscribe to best effort information and switch over from point-to-point UDP to multicast UDP.

14.4 Operating over Firewalls

To have a federate (LRC) operating behind a firewall you will need to open some ports for incoming traffic from other federates. A stateful packet inspection firewall (or router filters) is assumed. It is also assumed that you only have one federate per host.

1. For N federates, start by limiting the port range to N+1 TCP ports and one UDP port. Assign the same port range to all federates on all hosts.
2. Open the specific port ranges (TCP and UDP) for incoming connections from the IP addresses of the other federates.
3. The above has to be applied for each federate behind a firewall.
4. Avoid using multicast.

A federate that is number M to start will actually use M+1 incoming TCP ports and one incoming UDP port. You are recommended not to rely on the start order when configuring firewalls.

You may also consider establishing an encrypted VPN between sites.

To run the CRC behind a firewall you will need to open the *CRC Master port* in the firewall for incoming TCP connections from all federate addresses.

14.5 Network Settings

All network settings for the CRC are accessible from the CRC Settings editor application as shown in Figure 58.

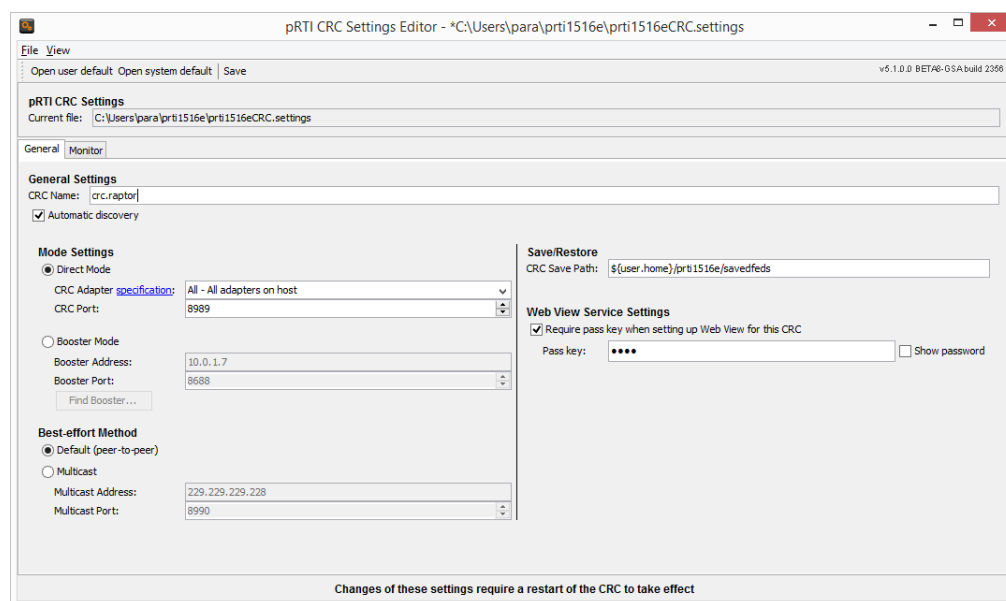


Figure 58 – Changing the settings for the CRC.

See section 16.1 for more information on changing the CRC settings.

For the LRC the network settings are available in both the LRC GUI shown in Figure 59 and in a settings file.

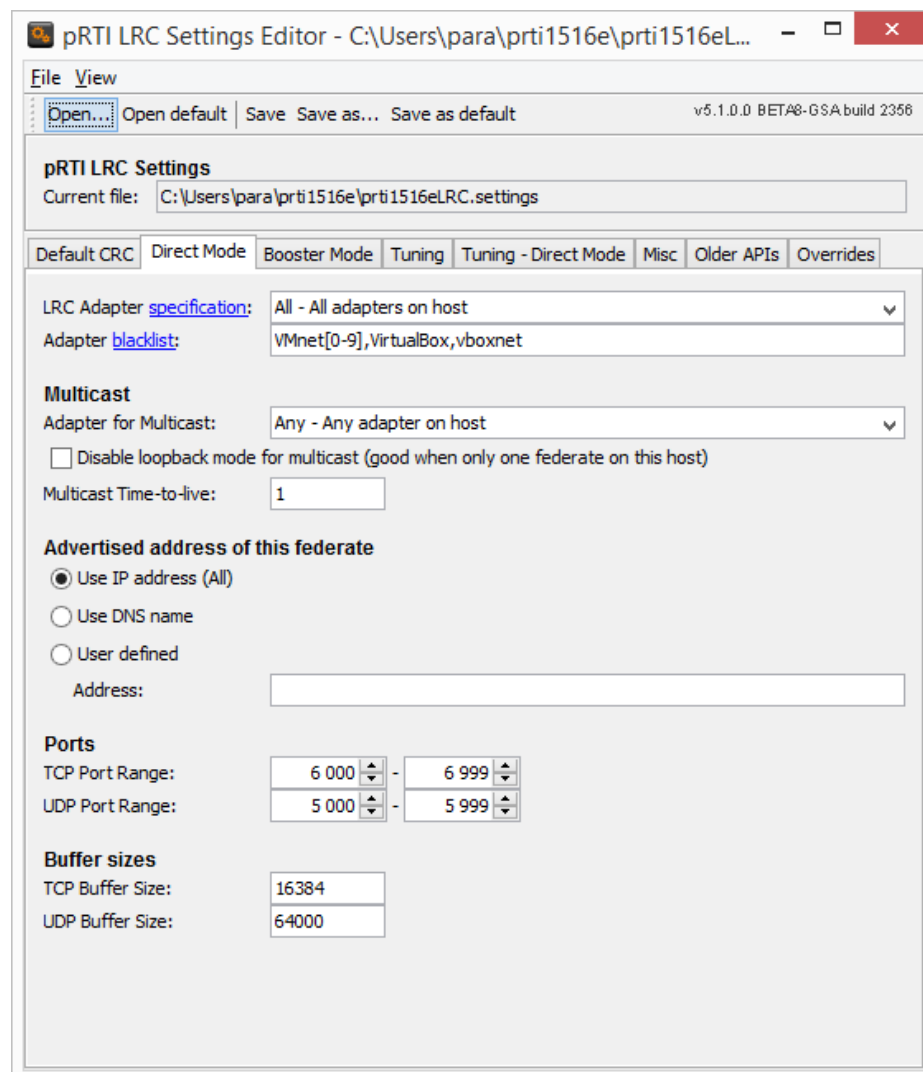


Figure 59 – Changing the settings for the LRC.

See section 16.2 for more information on changing the LRC settings.

14.6 Pitch pRTI™ and Pitch Booster™

By default Pitch pRTI™ offers excellent performance on a LAN (Local Area Network). For WAN (Wide Area Network) such as the Internet or a corporate network, the capacity (bandwidth, latency) is more limited than on a LAN. To be able to offer the best possible performance for WAN an additional component for Pitch pRTI™ is available: the Pitch Booster™. The Pitch Booster™ is a separately licensed product.

The normal configuration is as follows. One Pitch Booster™ is configured for each site.

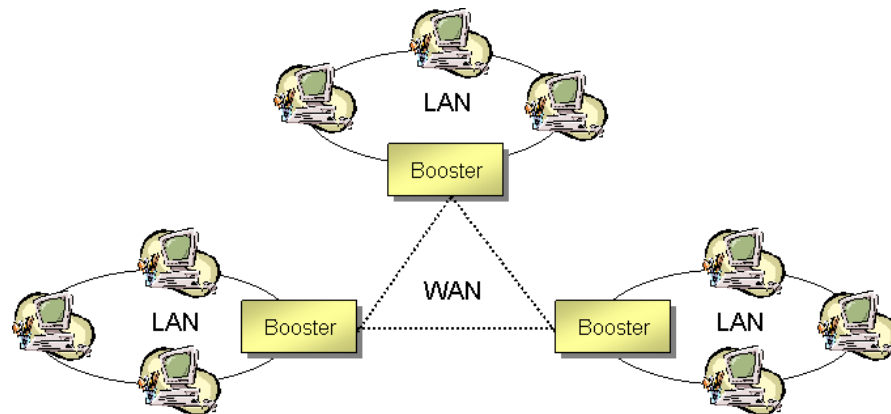


Figure 60 – The Pitch Booster™ topology.

The Pitch Booster™ provides a number of advantages:

1. Easier to run simulation between different sites, especially across firewalls.
2. Run several RTIs and several federations at the same time.
3. Discover RTIs and federates between several sites. It doesn't matter where you run the central RTI component.
4. Improved bandwidth usage, performance and scalability. Data sent through the Pitch Booster™ is concentrated on the sending side and exploded on the receiving side.
5. Improved reliability. Less "best-effort" data will be lost.
6. Less work on the federate side. The work with sending updates will be handled by the Pitch Booster™ on behalf of the federates.
7. Improved security handling. When running over an open WAN only the Booster needs to communicate over the WAN. This will reduce the firewall configuration required.

14.6.1 Configuring the LRC and CRC for Pitch Booster™

To enable the use of the Pitch Booster™, open the CRC network settings editor, go to the Booster tab, choose *Booster Mode* instead of *Direct Mode* and specify the address of the Booster. In case the host running Pitch Booster™ has several addresses it is necessary to specify an address belonging to the LAN subnet.

For the CRC, you must also specify a CRC nickname which is a name that is unique among all CRCs in your Booster network. This name will be used by federates in the Booster network to address your CRC, as the setting of the CRC-host property in the local settings designator. See Pitch Booster™ User's Guide for more information about CRC nicknames.

The LRC is told to use booster for communication by the way that CRC address is formatted. The format is: **<CRC-name>@<local-booster-address:port>**

So, connecting your federate to a CRC named "MyCRC" accessible through your booster network, and your local booster is running at 192.168.1.20:8688 can be done by formatting the CRC address as follows:

MyCRC@192.168.1.20:8688

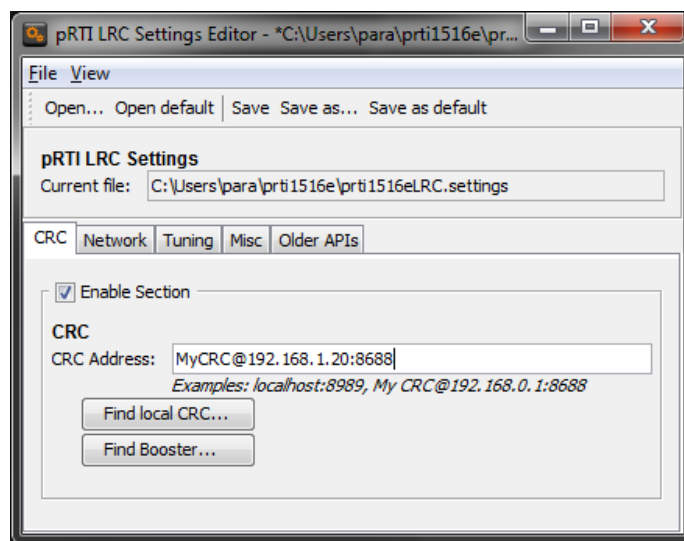
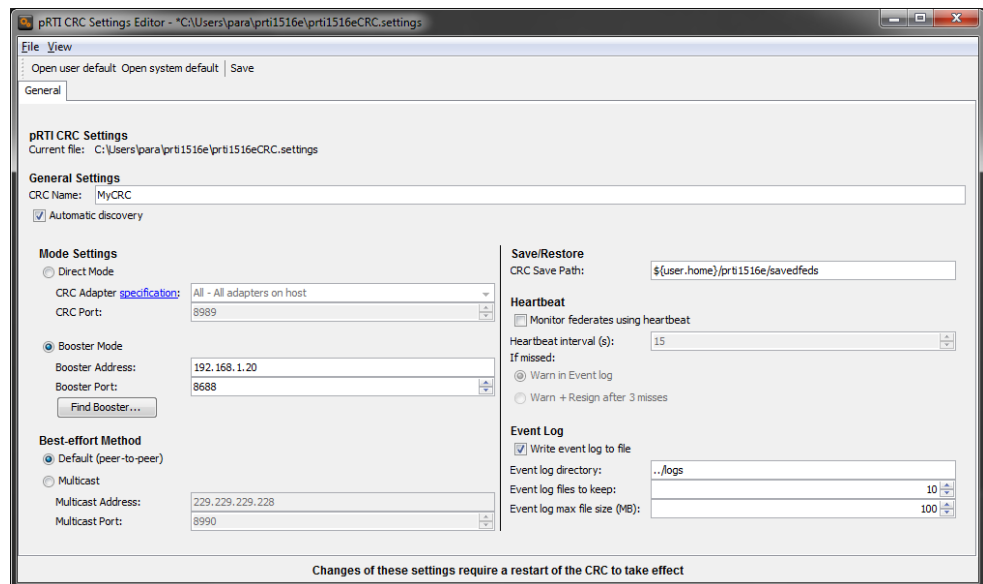


Figure 61 – Enabling Pitch Booster™ through the CRC and LRC Settings.

15 Advanced Pitch pRTI™ Network Performance Tuning

15.1 Introduction

By default, Pitch pRTI™ is tuned for minimal latency over local area networks with high capacity. This means that in many cases there is no need to tune Pitch pRTI™, especially during the early federation development phase. When you deploy your federation and scale up you are likely to get closer to the capacity of participating components such as networks, computers and federates. This chapter gives you more insights into the advanced tuning possibilities that Pitch pRTI™ offers.

There are several powerful ways to tune the network performance of Pitch pRTI™. Tuning means that you adapt the behavior of the RTI to meet the specific needs of your federation or to compensate for limitations in your network or federates. In many cases it also means that you are willing to trade one type of performance for another.

Some of the performance improvements that can be achieved by tuning are:

- Reduced latency, that is, the time it takes for an update to travel to another federate.
- Increased throughput, that is, the number bytes transmitted per second.
- Increased update rate, that is, the number of updates transmitted per second, which is very similar to the previous.
- Reduced loss for updates that use the *best-effort* transportation type.

You should try to get some type of performance metrics for your specific federation before you tune:

- In order to verify that you actually do have a performance problem.
- To compare with your predicted performance figures and
- In order to determine when your tuning affects the federation performance in the desired way.

15.2 Federate Tuning

RTI network tuning may be very useful but the performance of the federates may be just as important to make your federation run well. Two common situations where the RTI performance is reduced by federates are:

- A federate does not consume incoming messages as fast as the senders produce them. For best-effort updates these may be discarded. Some tuning parameters for this are described below. For reliable updates, interactions, object discoveries, etc, it is not acceptable to discard messages. This means that these messages will be queued and potentially slow down the sending federate and the entire federation.
- A federate spends a lot of time in each call to the *FederateAmbassador* (callbacks from the RTI). This prevents the RTI from delivering additional messages. This may in some cases reduce the speed both for other federates and the entire federation.

Both of these cases will usually result in a federate having a large queue of incoming messages. Use the Pitch pRTI™ *DUMP* command (issued in the Pitch pRTI™ console) to check the FIFO and TSO queue lengths if you suspect that you may have these problems.

15.3 Setting Tuning Parameters

There are two basic ways to set tuning parameters:

1. Using presets: There are a number of predefined settings for common situations. By simply selecting the preset that most closely matches your needs you may improve the performance.
2. Advanced tuning: You can also adjust each parameter individually. Detailed instructions for this are provided below. We recommend that you use one of the presets as a starting point for this type of tuning.

Before you start tuning it is useful to understand that Pitch pRTI™ is tuned by default and that the default tuning is for minimal latency (i.e. minimum delay between federates).

15.3.1 Using Presets

You can tune each federate individually based on:

- The optimization criteria (latency, loss rate, etc).
- How it produces and consumes information from other federates.
- Network properties.

To tune, simply ensure that the federate is not running, open the *LRC Settings* GUI and select the *Tuning* tab as shown in Figure 62.

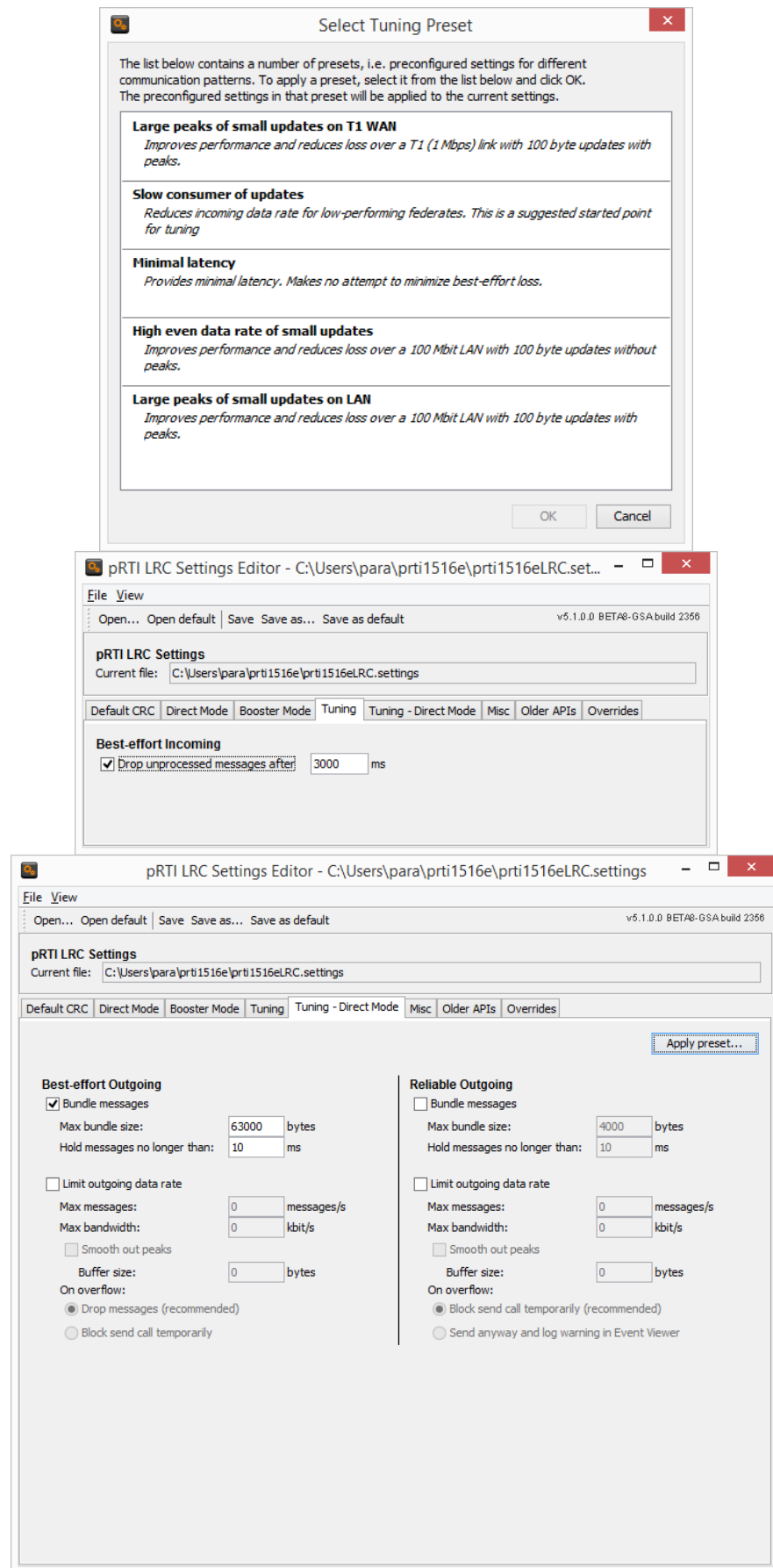


Figure 62 – The tuning settings in the LRC Settings GUI.

To select a preset simply select it in the tuning preset pop-up window. The predefined presets included with Pitch pRTI™ are listed in Table 1.

Preset Name	Description
Minimal latency (default)	Provides minimal latency. Makes no attempt to minimize best-effort loss
Large peaks of small updates on T1 WAN	Improves performance and reduces loss over a T1 (1 Mbps) link with 100 byte updates with peaks.
Large peaks of small updates on LAN	Improves performance and reduces loss over a 100 Mbit LAN with 100 byte updates with peaks.
High even data rate of small updates	Improves performance and reduces loss over a 100 Mbit LAN with 100 byte updates without peaks.
Slow consumer of updates	Reduces incoming data rate for low-performing federates

Table 1 – Available Pitch pRTI™ tuning presets.

Each preset is stored in a separate file in the *prti1516e* subdirectory of the user home directory (e.g. *C:\Documents and Settings\username\prti1516e* on Windows). Advanced users may define additional presets and add to this directory.

15.3.2 Advanced Tuning

Before you do advanced tuning you need to know some characteristics of your federation. Following are some of the most important questions that you need to answer to do effective tuning.

General Strategy

1. Is some latency acceptable in order to reduce best effort loss or increase throughput for best-effort and reliable? (y/n)
2. Is this federate expected to consume incoming updates at a significant slower rate than senders will produce updates? (y/n)
3. If this federate sends best effort data faster than other federates can receive, how do you want to handle this? Reduce this federates speed or discard data?
4. If this federate sends reliable data faster than other federates can receive, how do you want to handle this? Reduce this federates send speed or throw an exception for this federate.

Connection

1. Are you running over a WAN or LAN?
2. What is the bandwidth of the LAN? (10/100/1000 Mbit/s)
3. What is the bandwidth of the WAN? (x kb/s)

Outgoing Best-Effort Data

1. What is the typical size for best effort data? (bytes)
2. What is the typical rate for best effort data? (Hz)

3. Do you have peaks in the update rate from time to time? (y/n)

Outgoing Reliable Data

1. What is the typical size for reliable data? (bytes)
2. What is the typical rate for reliable data? (Hz)

Incoming Best-Effort Data

1. Is it acceptable to discard older updates in order to keep up with newer updates? (y/n)
2. After what time can an update be considered old enough to be discarded?

Based on your answers to these questions you can determine the parameters of the network tuning algorithms that pRTI™ 1516 offers.

15.4 Pitch pRTI™ Tuning Algorithms

Figure 63 describes the chain of tuning algorithms used by Pitch pRTI™.

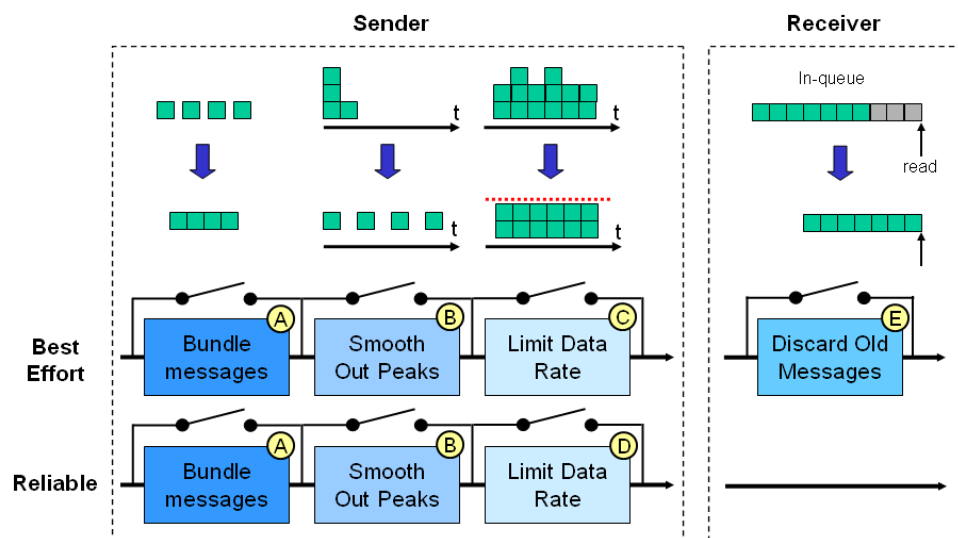


Figure 63 – Pitch pRTI™ tuning algorithms.

Note that the order between the algorithms is fixed but you can choose which ones that you want to use and which ones that you want to bypass. Below is a detailed description of each of these algorithms referred to as A to E in Figure 63.

15.4.1 Algorithm A – Bundling

Instead of sending each update individually the RTI may wait for more messages and bundle them. You may want to specify how long the RTI should wait and a limit for the size of the bundle.

Pros: Increased throughput. Also reduces loss for best-effort.

Cons: Increased latency, up to the wait time value.

Table 2 describes the parameters which the bundling algorithms requires.

Parameter Name	How To Calculate
Max bundle size (byte)	Depends on the type of link used. Traditional Ethernet and WAN links often have 1500 bytes. Gigabit with Ethernet has 9000 bytes. Best-effort on LAN should not exceed 64000 bytes. Slow serial links may have less than 1500. Advanced users may also want to examine the Maximum Transmission Unit (MTU) using the operating system command "ping -f -l <size> <host>".
Max hold time (ms)	Lower than 1/Update rate (Hz). To compensate for unsynchronized federates you may want to use less than half of the time. For a 50 Hz update rate this means less than half of 20 ms.

Table 2 – Bundling parameters.

15.4.2 Algorithm B and C – Limit Bandwidth Usage

These algorithms limit the outgoing message rate by smoothing out peaks and either limiting the updates produced or dropping outgoing best-effort messages. See section 15.4.3 and 15.4.4 for detailed descriptions of these two algorithms.

Table 3 lists the parameters required by these algorithms to limit the data rate. The algorithms will limit the rate based on the lower of these two values.

Parameter Name	How To Calculate
Max messages	If there are limitations on how many updates receiving federate can consume, use the highest value for this parameter.
Max bandwidth	If there are limitations on the bandwidth that this federate can use then use this parameter.

Table 3 – Throttling parameters.

15.4.3 Algorithm B – Smooth Out Peaks

Some federates produce updates for all of their objects at a certain point in time. This may result in a large number of updates in a short time. Especially for best-effort this may be a problem since the network may discard a large amount of the messages. For both best-effort and reliable it may result in large temporary loads on receiving federates.

The bandwidth is determined by the parameters listed in Table 3. The buffer size parameter listed in Table 4 can be determined by the user. It is important that the average data flow from the federate does not exceed the maximum values specified by the parameters listed in Table 3. Otherwise the buffer will be filled and algorithm C or D will be activated.

Pros: Increased throughput. Also reduces loss for best-effort.

Cons: Increased latency.

Parameter Name	How To Calculate
Buffer size	Bigger than the time interval between peaks multiplied by the average bandwidth (including the peaks).

Table 4 – Smoothing parameters.

15.4.4 Algorithm C – Drop Best-Effort Messages or Block Sender

If the desired data rate cannot be achieved then Pitch pRTI™ needs to reduce the data rate. For best-effort this can be done by simple discarding the data. Another option is to prevent the federate from producing more data by simply not returning from the send call until the data has been sent. This will prevent the federate temporarily from making additional send calls.

Pros: Reduces data flow. Reduces the load for other federates.

Cons: Data loss or federate slowed down.

15.4.5 Algorithm D – Block Sending of Reliable Messages or Produce Warning

If the desired data rate cannot be achieved then Pitch pRTI™ needs to reduce the data rate. For reliable data this can be done by preventing the federate from producing more data by simply not returning from the send call until the data has been sent. This will prevent the federate temporarily from making additional send calls. Another option is to exceed the data rate and send a warning to the event log. This may be used as input for modifying the design of the federate or the federation.

Pros: Reduces data flow. Reduces the load for other federates.

Cons: Federate may be slowed down.

15.4.6 Algorithm E – Drop Old Incoming Best-Effort Messages

If a federate consumes data at a slower rate than it arrives, it will start lagging behind. At some point in time the memory will eventually run out. In many cases it makes no sense for a federate to consume older updates when newer updates have already arrived. The RTI can discard older data. This technique can only be applied for best-effort data.

Pros: Reduced federate load.

Cons: Data loss.

Table 5 lists the parameters used by this algorithm.

Parameter Name	How To Calculate
----------------	------------------

Parameter Name	How To Calculate
Drop unprocessed messages after x seconds	This parameter should be larger than the time between incoming updates. You may for example set this parameter to 2 – 3 times higher. A lower number will make the federate more responsive but increase the risk that the federate does not get any data at all.

Table 5 – Old incoming message discarding parameters.

16 Configuration Reference

This section contains a list of all the configuration switches that can be set in Pitch pRTI™. Normally, you do not need to use any of these configuration switches. They are intended for advanced users.

16.1 Settings for the Central RTI Component

The settings for the CRC can be modified from the pRTI™ Explorer user interface shown in Figure 64.

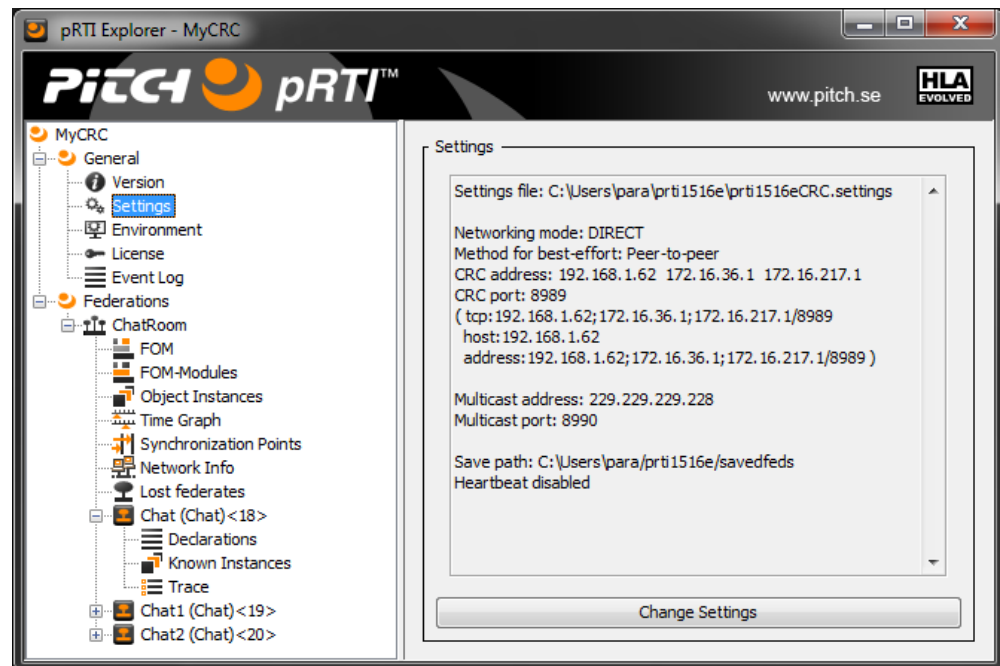


Figure 64 – The CRC specific settings.

Click the *Change Settings* button to modify the settings. Make the necessary changes and then restart Pitch pRTI™.

Note that all the settings are saved to a file called *pRTI1516eCRC.settings*. This file is located in the *prti1516e* directory located in your user home directory (e.g. *C:\Documents and Settings\username\prti1516e* on Windows). This is where the settings are saved when clicking "save as user settings" and these settings are used when running the CRC on the desktop as a logged in user.

When running the CRC in service mode, the settings will be read from a file with the same name (*pRTI1516eCRC.settings*) located in a system wide location. The system wide location is the pRTI installation directory on Windows, and */etc/prti1516e* on Linux and Mac OS X. This is where the settings are being saved when clicking "save as system settings".

If you wish to use a *pRTI1516eCRC.settings* file located elsewhere, e.g. if you wish to run multiple CRC:s on the same computer you can use the Java property *settings.dir* when starting the CRC as shown below, and specify it in the *.vmoptions* file (*pRTI1516e.vmoptions* for the CRC running on the desktop and *pRTI1516e-service.vmoptions* for the CRC running in service mode, both located in the *bin* subdirectory of the installation).

-Dsettings.dir=c:\mydir

The CRC will then look for the *pRTI1516eCRC.settings* file in the *c:\mydir* directory instead.

The CRC settings editor can be found on the Windows start menu, or in the *bin* sub directory of your Pitch pRTI™ installation.

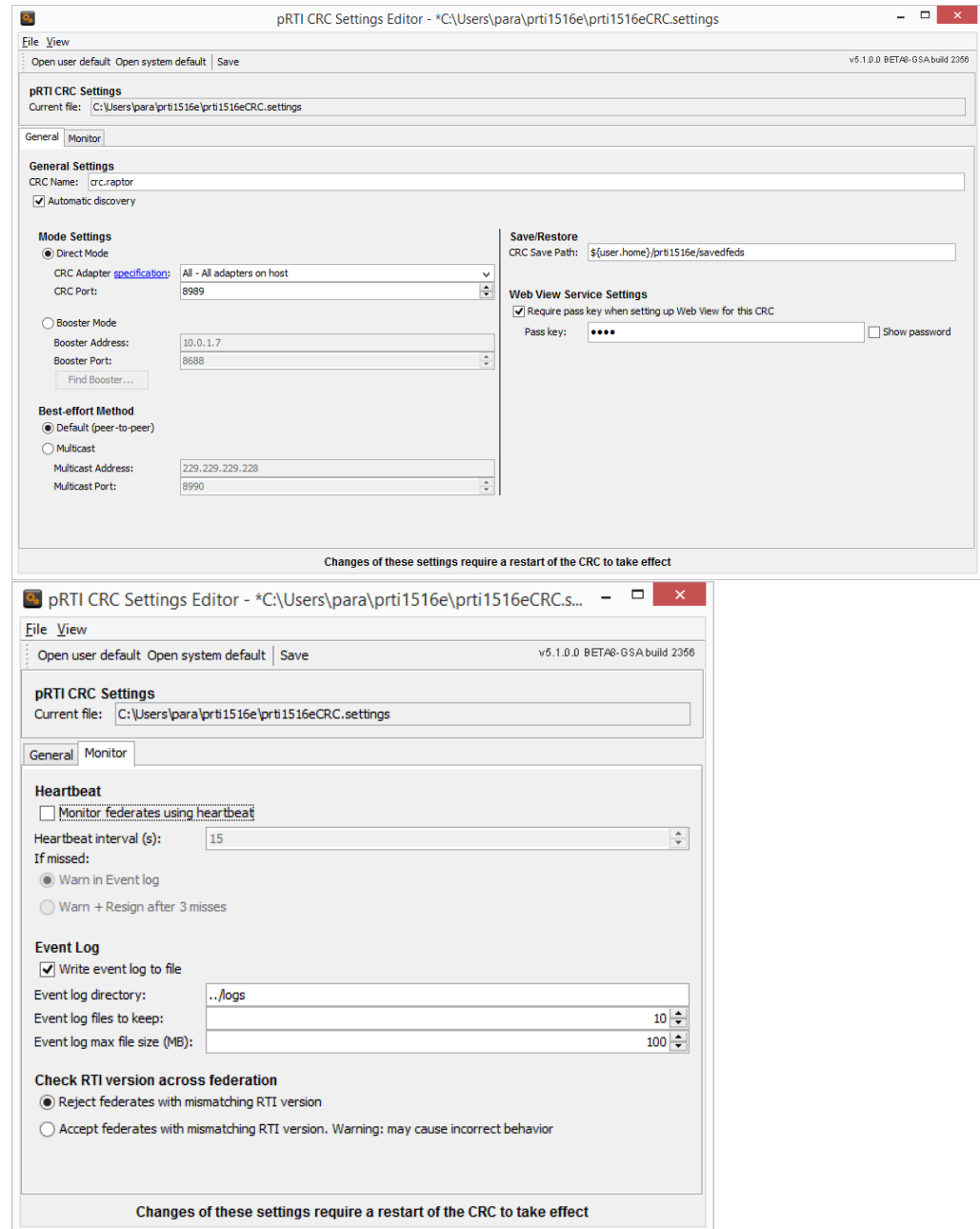


Figure 65 – The CRC settings editor.

Table 6 describes the setting available for the CRC.

Parameter name	Description
----------------	-------------

Parameter name	Description
CRC Name	A name for the CRC. This name is used to identify your CRC when it is being automatically discovered, but more importantly used as a unique CRC identifier when connecting the CRC to a booster network.
Automatic discovery	This option enables the CRC to be automatically located by other applications on your network. The LRC settings editor is an example of an application using this feature.
Mode Settings	Option for setting if communication through a booster should be used or not. If using a booster, the booster address and port are to be specified. Local boosters can be automatically detected by clicking the "Find booster"-button. If not using a booster, "direct mode" is specified. The port to use, and the interface(s) to use can be specified here.
CRC Adapter	The IP address(es) where the CRC is listening to incoming connections from federates. Adapter specification can be an exact or partial match for adapter IP, adapter name, or adapter description. A partial match must be unique.
CRC Port	The port number that the CRC uses to listen for incoming connections from federates.
Booster Address	The IP address to the local booster used in Booster Mode
Booster Port	The port number on the local booster used in Booster Mode
Best-effort Method	Specified whether or not to use multicast to deliver best effort messages.
Multicast Address	The multicast address used by the CRC.
Multicast Port	The port number that the CRC uses to listen for incoming connections from federates when using multicast.

Parameter name	Description
CRC Save Path	The directory where the CRC stores information when the save and restore services are used in the RTI. \${user.home } resolves to the current user's home directory.
Web View Service pass key	An option to set a passkey required by Web View server to be able to access the CRC.
Heartbeat/Monitor federates using heartbeat	Check to make the CRC monitor federates using heartbeat messaging with the LRCs. The purpose is to detect and optionally automatically resign federates whose process is not responding or hanging.
Heartbeat interval	The number of seconds between heartbeat messages.
If missed	Action to take when an LRC does not respond to heartbeat messages. The options are: Warn in Event log – which is only a warning and does not have any effect on the continuation of the federation execution. Warn + Resign after 3 misses – When an LRC has not responded to 3 heartbeat messages it is being automatically resigned from the federation by the CRC.
Write event log to file	Check to write the CRC event log to file in addition to displaying it in pRTI Explorer.
Event log directory	Directory where to store event logs written to file
Event log files to keep	The maximum number of event log files to keep when writing the event log to file
Event log max file size	The maximum size that an event log file may grow to before it is being rotated to a new file.
Check RTI version across federation	Options for how to handle connecting LRCs with a mismatching version. Either reject or accept with a warning.

Table 6 – The CRC settings.

16.2 Settings for the Local RTI Component

The settings for the LRC on each federate host are modified from a graphical user interface. To open the settings, use the command *LRC Settings* from the Start menu. The LRC settings can be divided up in four categories:

- Default CRC
- Direct Mode network settings
- Booster Mode network settings
- Tuning settings
- Tuning for Direct Mode settings
- Misc Settings
- Older APIs settings
- Overrides settings

These sections provide a reference summary of all the settings that are available but the details are described throughout this document. See chapters 14 and 15.

The LRC settings are saved in the *pRTI1516eLRC.settings* file in the *pRTI1516e* directory located in your home directory. By default, all federates running on the same computer, under the same user, use the same LRC settings.

16.2.1 Local Settings Designator files

If you want to use a special LRC setting for one federate, you may specify a *Local Settings Designator* in the call to the RTI ambassador method "connect" in your federate code. The local settings designator is an abstract reference to a set of LRC settings to be used for overriding the default settings mentioned above.

The local settings designator file (.lsd) is used for overloading some or all of the properties in the LRC settings file. Therefore the same LRC settings editor is being used for editing .lsd files. When saving a file, you may choose to save it as .lsd. Since the lsd file may only contain certain properties of the whole set of LRC settings, there are checkboxes (labeled "enable section") for each section which is used to set whether a set of properties are to be set in the .lsd file or not.

When specifying a local settings designator, such as "MySpecialSettings" in the call to connect, the LRC will search for additional settings as follows:

First, it will look for a file named "MySpecialSettings.lsd" (note the file suffix) in the in a **system wide location**. The system wide location is in the pRTI installation directory on Windows, and in */etc/pRTI1516e* on Linux and Mac OS X.

Then it will look for a file named "MySpecialSettings.lsd" in the **home directory of the current user**.

Then it will look for a file named "MySpecialSettings.lsd" in the **working directory** of the federate.

Any settings that it manages to read from the file in those three locations will be overriding the default settings, and in case there are settings available in several of these locations they will override each other in the same order as the files are read.

If no abstract local settings designator name is specified, the LRC will look for a file named "default.lsd", in the same search order as above.

16.2.2 Advanced override of local settings

There are methods for doing an external override of the LRC settings and LSDs used by your federates. This method is called the "alternate settings" and can

override everything specified at all other levels (LSD-string in connect call, .lsd files, LRC settings file).

The alternate settings are settings parameters stored in a .lsd file just as other special settings are.

The RTI will search for alternate settings in the following order:

1. If there is a file named "alternate.lsd" in the working directory of the federate, the alternate settings from that file will be used.
2. If the environment variable *alternate_lsd* exists, its value will be used as LSD.
3. If the Java system property *alternate_lsd* exists, its value will be used as LSD.

16.2.3 Default CRC

Figure 66 shows the Default CRC settings tab in the *LRC Settings editor* and Table 7 provides a summary of the settings available.

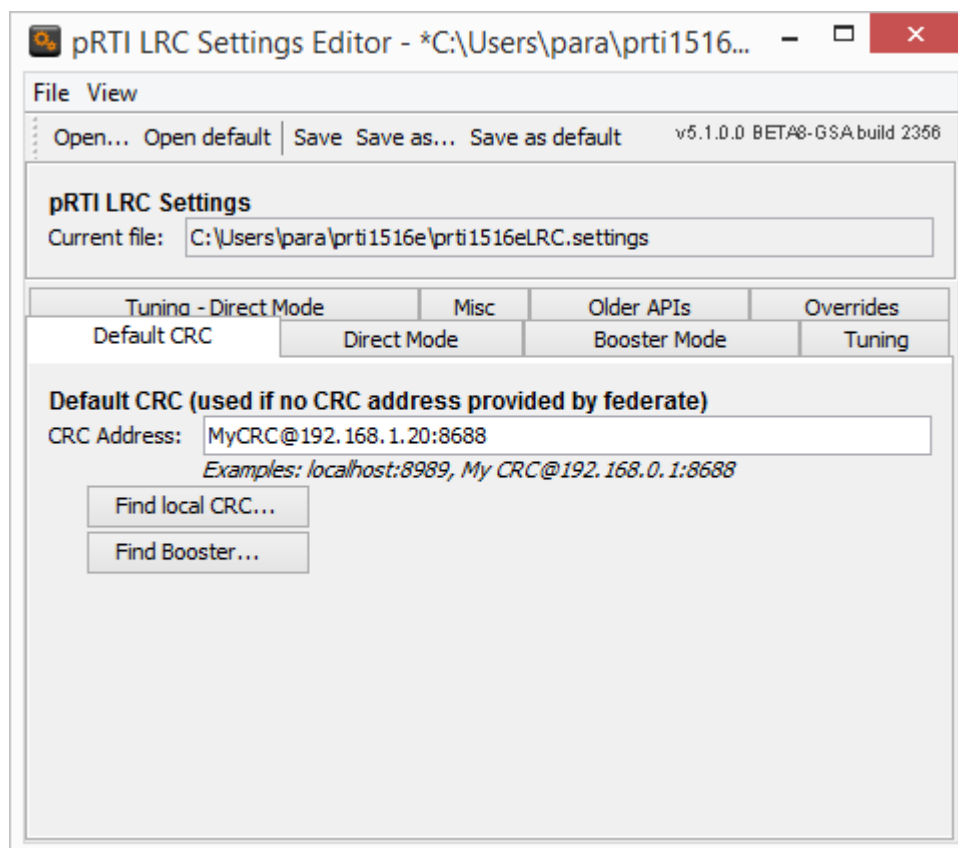


Figure 66 – The Default CRC settings tab.

General Parameters	Description
--------------------	-------------

General Parameters	Description
CRC-address	<p>The CRC address can optionally be specified here. The CRC address can also be specified directly in the local settings designator ("crcHost=..."). This is the way that versions prior to v 4.4 has required the address to be set. With v 4.4 and later this can be specified in this LRC settings field instead, so that your federate doesn't need to handle this.</p> <p>Depending on how the CRC-address is formatted, a direct network connection mode or a booster connection mode will be used.</p> <p>By setting the CRC-address to <i>hostname:port</i> or to <i>IP-address:port</i> you will use a regular, direct network connection which is the case when not using boosters at all.</p> <p>By setting the CRC-address to <i>CRC-name@booster-LAN-IP:port</i> you will use a connection through a booster network</p> <p>The button "Find local CRC" will automatically try to find a CRC on your local network for direct connections.</p> <p>The button "Find Booster" will be able to help you locate the local address to boosters on your network.</p>

Table 7 – CRC LRC settings.

16.2.4 Direct Mode network Settings

Figure 67 shows the *Direct Mode* settings tab in the *LRC Settings editor* and Table 8 provides a summary of the settings available. For a deeper discussion on network configuration see chapter 14.

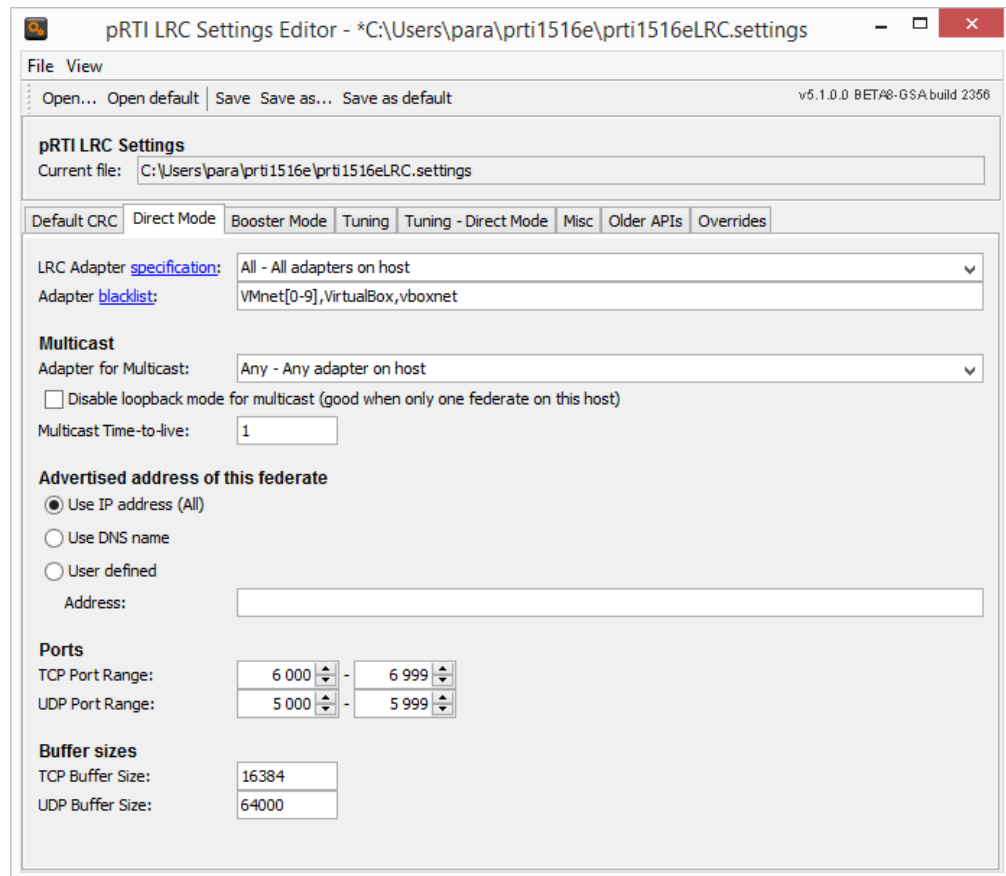


Figure 67 – The LRC Settings Direct Mode tab.

Network Parameters	Description
LRC Adapter	Which IP address/addresses the LRC is using for network communication. Can be set to <i>All</i> , a single IP address or a specified pattern. Adapter specification can be an exact or partial match for adapter IP, adapter name, or adapter description. A partial match must be unique.
Adapter for Multicast	Which network adapter to use when sending multicast traffic. Must one and only one adapter, i.e. <i>All</i> is not allowed as it is with the <i>LRC.adapter</i> setting.
Advertised address of this federate	This setting can be used to enforce the advertisement of a specific IP-address. When using the “Use IP address” option, a <i>black list</i> of IP-addresses, interface names or patterns thereof can be defined to avoid advertising the address of certain interfaces.

Network Parameters	Description
TCP Port Range	The port range used by the LRC for connecting to either the CRC or different LRC:s.
UDP Port Range	The same as <i>TCP Port Range</i> but for UDP.
TCP Buffer Size	You may set the TCP buffer size that this LRC will use. Note that this setting is only a guiding setting; the buffer size is not guaranteed to be set to the size that you specify.
UDP Buffer Size	You may set the UDP buffer size that this LRC will use. Note that this setting is only a guiding setting; the buffer size is not guaranteed to be set to the size that you specify.

Table 8 – LRC Direct Mode network settings.

16.2.5 Booster Mode network Settings

Figure 68 shows the *Booster Mode* settings tab in the *LRC Settings editor* and Table 9 provides a summary of the settings available. For a deeper discussion on network configuration see chapter 14.

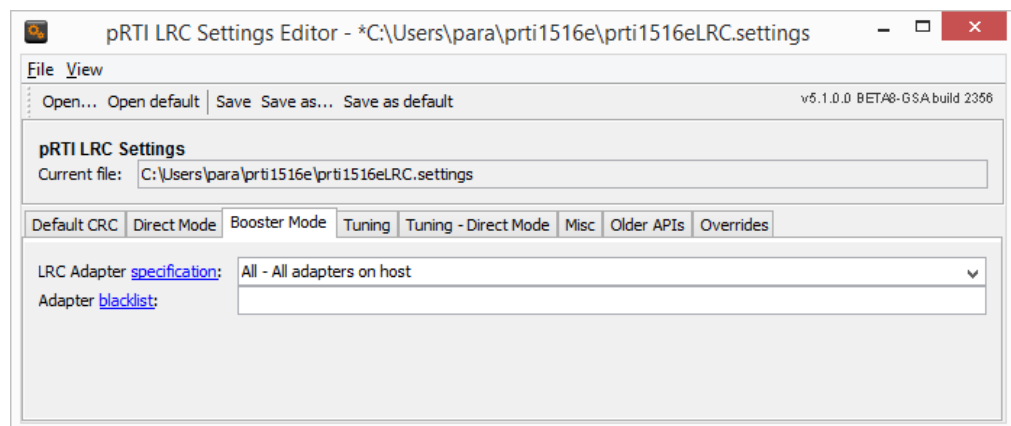


Figure 68 – The LRC Settings Booster Mode tab.

Network Parameters	Description
LRC Adapter	Which IP address/addresses the LRC is using for network communication. Can be set to <i>All</i> , a single IP address or a specified pattern. Adapter specification can be an exact or partial match for adapter IP, adapter name, or adapter description. A partial match must be unique.

Table 9 – LRC network settings.

16.2.6 Tuning Settings

Figure 1 shows the general *Tuning* settings tab in the *LRC Settings editor* and Table 10 provides a summary of the settings available (more tuning settings for Direct Mode can be done in the *Tuning – Direct Mode* tab). For a deeper discussion on network tuning see chapter 15.

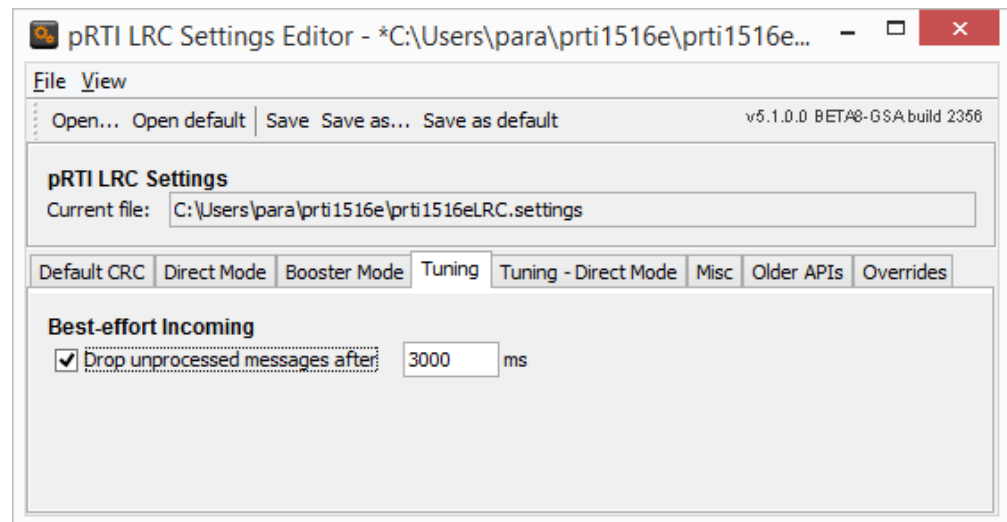


Figure 69 – The LRC Settings Tuning tab.

Best-effort Tuning Parameters	Description
Drop unprocessed messages after x ms	If enabled, old messages waiting to be delivered to the federate will be discarded when they have reached the ages specified.

Table 10 – LRC tuning settings.

16.2.7 Tuning – Direct Mode Settings

Figure 70 shows the *Tuning – Direct Mode* settings tab in the *LRC Settings editor* and Table 11 provides a summary of the settings available. For a deeper discussion on network tuning see chapter 15.

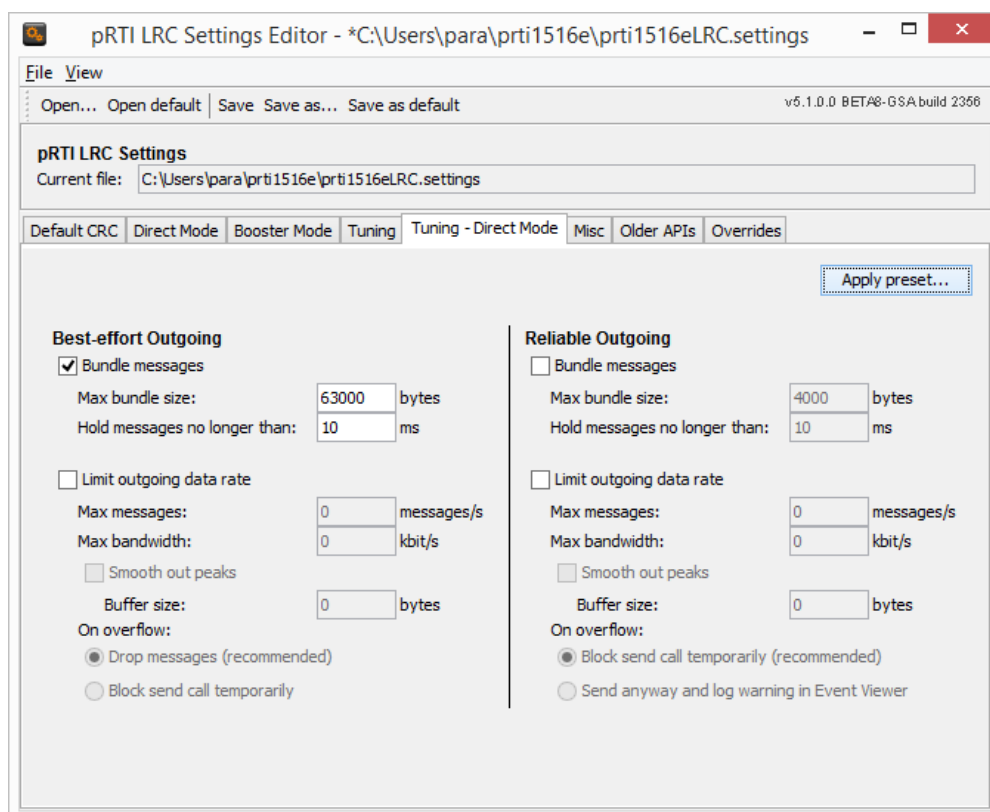


Figure 70 – The LRC Settings Tuning – Direct Mode tab.

Best-effort Tuning Parameters	Description
Bundle messages	If enabled the RTI will bundle multiple messages before delivering. See section 15.4.1 for details.
Max bundle size	The maximum size of a bundle when <i>Bundle messages</i> is checked.
Hold message no longer than x ms	The longest time to wait before sending a bundle, regardless of its size.
Limit outgoing data rate	If enabled, the RTI will not send messages at a rate higher than specified by the following two parameters. See section 15.4.2 for more details.
Max. messages	The maximum number of messages to send per second when <i>Limit outgoing data rate</i> is checked.
Max bandwidth	The maximum send rate (in bits/s) produced by the RTI when <i>Limit outgoing data rate</i> is checked.
Smooth out peaks	Smooths out large peaks of data if <i>Limit outgoing data rate</i> is checked. See section 15.4.3 for more details.
Buffer size	The buffer size used when <i>Smooth out peaks</i> is checked.
On overflow	Specifies what to do when the buffer is full and <i>Limit outgoing data rate</i> is enabled.
Drop unprocessed messages after x ms	If enabled, old messages waiting to be delivered to the federate will be discarded when they have reached the ages specified.
Reliable Tuning Parameters	Description
Bundle messages	If enabled the RTI will bundle multiple messages before delivering. See section 15.4.1 for details.
Max bundle size	The maximum size of a bundle when <i>Bundle messages</i> is checked.
Hold messages no longer than x ms	The longest time to wait before sending a bundle, regardless of its size.
Limit outgoing data rate	If enabled, the RTI will not send messages at a rate higher than specified by the following two parameters. See section 15.4.2 for more details.
Max. messages	The maximum number of messages to send per second when <i>Limit outgoing data rate</i> is checked.
Max. bandwidth	The maximum send rate (in bits/s) produced by the RTI when <i>Limit outgoing data rate</i> is checked.

Best-effort Tuning Parameters	Description
Smooth out peaks	Smooths out large peaks of data if <i>Limit outgoing data rate</i> is checked. See section 15.4.3 for more details.
Buffer size	The buffer size used when <i>Smooth out peaks</i> is checked.
On overflow	Specifies what to do when the buffer is full and <i>Limit outgoing data rate</i> is enabled.

Table 11 – LRC Tuning – Direct Mode settings.

16.2.8 Misc Settings

Figure 71 shows the *Misc* settings tab in the *LRC Settings editor* and provides a summary of the settings available.

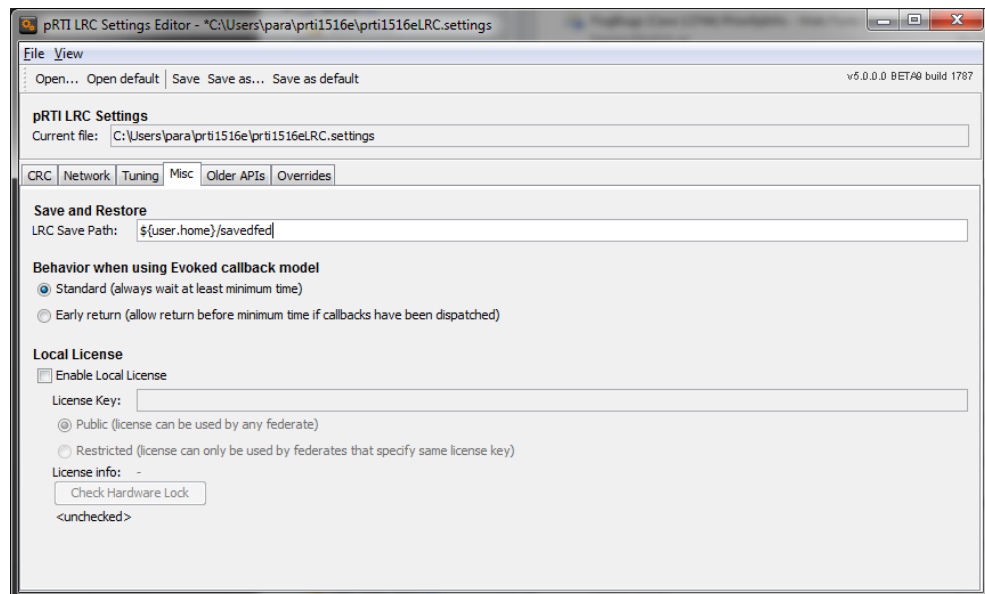


Figure 71 – The LRC Settings Misc tab.

Misc Parameters	Description
LRC Save Path	The path used for save and restore
Behaviour when using Evoked callback model	The way to handle federates with the HLA_EVOKED process model. The standard setting will always wait for callbacks at least the minimum time specified in the evoke-call. The early return setting will allow the evoke call to return before the minimum time has elapsed, as an optimization.
Enable local license	Check if using a local LRC license (see section 3.2.2 for further explanation)
License key	A license activation key for a local LRC license
Public OR Restricted	A public license can be used by any federate if there are seats left. A restricted license can only be used by federates using the same license key, if there are seats left

Table 12 – LRC Misc settings.

16.3 Older APIs settings

Figure 72 – The LRC Settings Older APIs tab shows the *Older APIs* settings tab in the *LRC Settings editor* and provides a summary of the settings available.

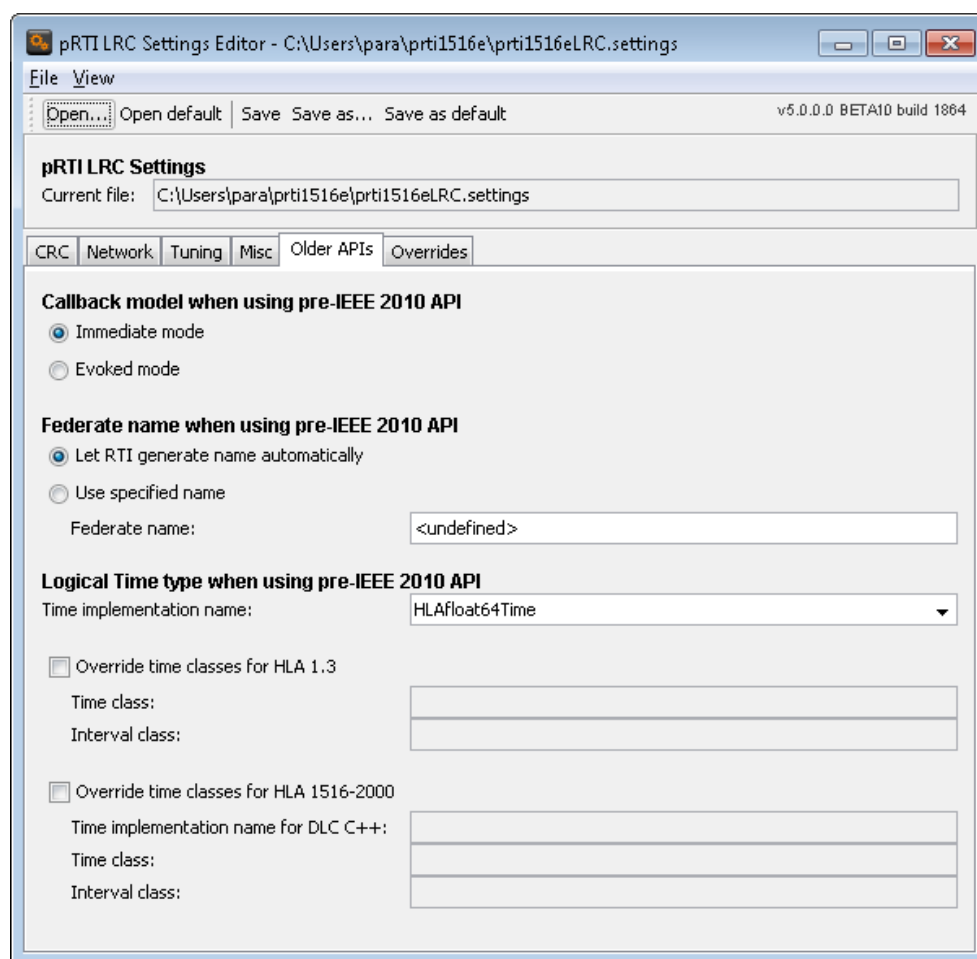


Figure 72 – The LRC Settings Older APIs tab

Older APIs Parameters	Description
Callback model when using pre-IEEE 1516-2010 API	Callback model used by legacy federates. Immediate or Evoked a.k.a. "ticked" or "single threaded"
Federate name when using pre-IEEE 1516-2010 API	Older APIs do not have the concept of federate name, and therefore it needs to be constructed for older API federates. Options are: Let RTI generate name automatically which will use the federate type and generate a name based on that. Use specified name which will set the federate name to the entered value.
Time implementation name	The IEEE 1516-2010 standard time representation to use for older APIs federates.
Override time classes for HLA 1.3	Time class implementation for user defined time classes used by HLA 1.3 federates
Override time classes for HLA 1516-2000	Time class implementation for user defined time classes used by HLA 1516-2000 federates

16.4 Overrides settings

Figure 73 – The LRC Settings Overrides tab shows the *Override* settings tab in the *LRC Settings editor*. It consists of a number of properties which normally are set by the federate through API calls to the RTI. The values from those API calls can be overridden by enabling the override of these properties and supply a value through LRC Settings.

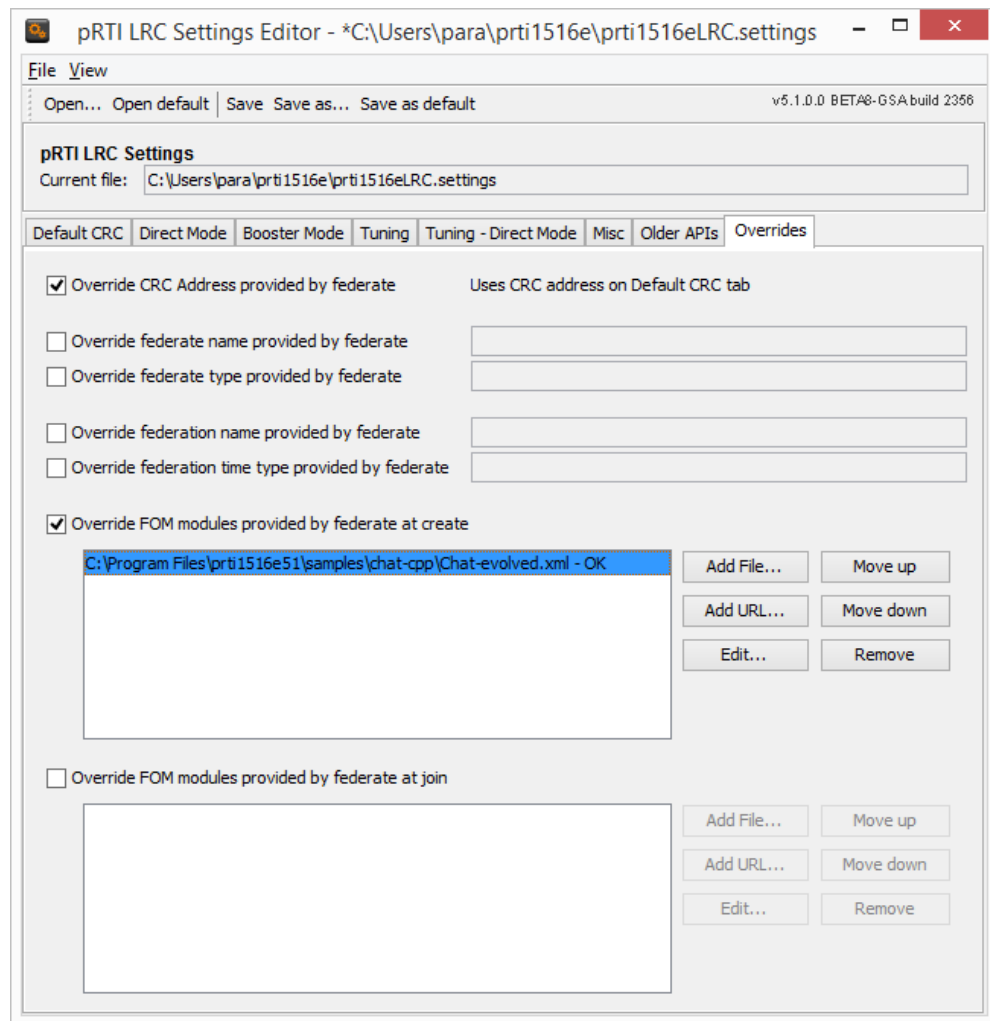


Figure 73 – The LRC Settings Overrides tab

16.5 LRC JVM Settings For C++ Federates

The environment variables `PRTI1516_OPTIONX` (where X is a number between 1 and 20) are used to pass parameters to the Java Runtime Environment when using a C++ federate. To specify the maximum amount of memory (in this example 256Mbytes) that the Java virtual machine used by the LRC is allowed to allocate, you would set a `PRTI1516_OPTION` variable like this:

```
set PRTI1516_OPTION2=-Xmx256m
```

See the *java* command for more information about the parameters that can be specified.

Starting with pRTI™ version 4.4, it is also possible to use a *.vmoptions* file for injecting JVM options. This is done by creating a text file named *prti.vmoptions* in the working directory of your federate. Each line in the *.vmoptions* file may contain one JVM option, so make sure to put each option per line in the file.

17 Using federates developed for Legacy HLA versions

Pitch pRTI™ comes with backwards compatibility libraries for supporting federates developed for the previous HLA versions, IEEE 1516-2000, IEEE 1516-2000 DLC and HLA 1.3 (see table below).

Starting with Pitch pRTI™ v 5, the support for the legacy standard versions is seamless and requires very little extra configuration overhead. Federates developed for different legacy standard versions can be mixed in the same federation, and there is no longer a need to convert monolithic FOMs of the old formats. Therefore Pitch pRTI™ v 5 is called “multi API compliant”.

Standard	Programming language	Comment
IEEE 1516-2010	C++	a.k.a. “HLA Evolved”
IEEE 1516-2010	Java	a.k.a. “HLA Evolved”
IEEE 1516-2000	C++	
IEEE 1516-2000	Java	
IEEE 1516-2000 DLC	C++	This is a variant of the C++ API for IEEE 1516-2000 that is designed for dynamic link compatibility.
HLA 1.3	C++	
HLA 1.3	Java	

To accomplish this, the installation structure has been slightly modified and the new structure is described in this section. The additional parameters needed to support the old APIs, such as choice of process model and logical time representation, are now controlled by LRC settings as described in section 16.2.

Pitch pRTI™ allows federates using all these API to co-exist in the same federation

17.1 C++ Libraries

When you install Pitch pRTI™, the C++ API libraries for all standards can be found in the *lib* subdirectory and the corresponding compiler specific subdirectory of your installation. Example: Let's say that your federate was developed for 32-bit Visual C++ 9.0. Then you should put `<pRTI install directory>\lib\vc90` on PATH, but you don't need to care about any API-version specific subdirectory. The *vc90* subdirectory contains all available API libraries for 32-bit Visual C++ 9.0.

17.2 Headers

The *include* subdirectory of the installation contains the IEEE 1516-2010 API headers, but it also contains the following subdirectories for legacy standard headers:

- *include/HLA1516-2000* contains header files for the IEEE 1516-2000 standard API.

- *include/HLA1516-2000DLC* contains header files for the IEEE 1516-2000 DLC API.
- *include/HLA13* contains the header files for the HLA 1.3 API.

17.3 Java libraries

The required Java library for each of the legacy standards is the same library as recognized from the corresponding older versions of Pitch pRTI™. The one used for IEEE 1516-2010 federates is *prti1516e.jar* in the *lib* directory of the Pitch pRTI™ installation. In that directory there is also *prti.jar* to be used with HLA 1.3 federates and *prti1516.jar* to be used with IEEE 1516-2000 federates.

Note that there are dependencies between the API library jar files and other Pitch pRTI™ implementation libraries, and keeping them in the same directory simplifies the CLASSPATH setting for federate users who should only care about putting the API library on CLASSPATH.

Note that the Java library need to be on the CLASSPATH also when running C++ federates.

17.4 Time classes

The different APIs use different time classes. To make sure that the federates can interoperate, Pitch pRTI™ provides automatic mapping between time classes in Multi-API federations.

Note that this is mainly a concern for federations using HLA Time Management Services.

The mappings are described in the tables below.

IEEE 1516-2010 time type	IEEE 1516-2000 DLC time type
HLAinteger64Time (default)	HLAinteger64Time
HLAinteger64Time	LogicalTimeImplInteger
HLAfloat64Time (default)	HLAfloat64Time
LogicalTimeDouble	LogicalTimeDouble

IEEE 1516-2010 time type	IEEE 1516-2000 time classes
HLAinteger64Time (default)	se.pitch.prti1516.time.HLAinteger64TimeFactory2000 se.pitch.prti1516.time.HLAinteger64TimeIntervalFactory2000
HLAinteger64Time	hla.time1516.LogicalTimeFactoryLong hla.time1516.LogicalTimeIntervalFactoryLong
HLAfloat64Time (default)	se.pitch.prti1516.time.HLAfloat64TimeFactory2000 se.pitch.prti1516.time.HLAfloat64TimeIntervalFactory2000
LogicalTimeDouble	se.pitch.prti1516.LogicalTimeDoubleFactory se.pitch.prti1516.LogicalTimeIntervalDoubleFactory

IEEE 1516-2010 time type	HLA 1.3 time classes
HLAinteger64Time (default)	se.pitch.prti.time.HLAinteger64TimeFactory13 se.pitch.prti.time.HLAinteger64TimeIntervalFactory13
HLAinteger64Time	se.pitch.prti.time.HLAinteger64TimeFactory13 se.pitch.prti.time.HLAinteger64TimeIntervalFactory13
HLAfloat64Time (default)	se.pitch.prti.time.HLAfloat64TimeFactory13 se.pitch.prti.time.HLAfloat64TimeIntervalFactory13
LogicalTimeDouble	se.pitch.prti.LogicalTimeFactoryDouble64 se.pitch.prti.LogicalTimeIntervalFactoryDouble64

Note that a single IEEE 1516-2010 time type, e.g. HLAinteger64Time, may be part of more than one mapping. The default mapping for each Evolved time type is marked in the tables.

Creating a federation

When a federation is created, it is assigned an IEEE 1516-2010 time type and, potentially, a IEEE 1516-2000 DLC time type. The IEEE 1516-2000 DLC time type is only assigned if the federation is created by a IEEE 1516-2000 DLC federate.

When an IEEE 1516-2010 federate creates a federation, it provides an IEEE 1516-2010 time type using the *timeImplementationName* parameter in the call to *createFederationExecution*.

When an IEEE 1516-2000 DLC federate creates a federation, it provides a IEEE 1516-2000 DLC time type using the *timeImplementationName* parameter in the call to *createFederationExecution*. The corresponding IEEE 1516-2010 time type is found in the time map.

When an IEEE 1516-2000 federate creates a federation, the RTI picks up an IEEE 1516-2010 time type from the LRC settings (LRC.logicalTimeForOlderApi setting).

When a HLA 1.3 federate creates a federation, the RTI picks up an IEEE 1516-2010 time type from the LRC settings (LRC.logicalTimeForOlderApi setting).

In all cases, the CRC uses the IEEE 1516-2010 time type and the *LogicalTimeFactoryFactory* to locate the corresponding IEEE 1516-2010 LogicalTimeFactory.

Joining a federation

When an IEEE 1516-2010 federate joins a federation, it uses the IEEE 1516-2010 time type of the federation and the LogicalTimeFactoryFactory to locate the corresponding IEEE 1516-2010 LogicalTimeFactory.

When a IEEE 1516-2000 DLC federate joins a federation, it uses the IEEE 1516-2000 DLC time type, if provided at create, otherwise the IEEE 1516-2010 time type is used to look up an IEEE 1516-2000 DLC time type. The IEEE 1516-2000 DLC time type is then used to look up matching IEEE 1516-2000 time classes in the time map. The IEEE 1516-2000 DLC time type is also used on the C++ side in a call to *LogicalTimeFactoryFactory::makeLogicalTimeFactory*. The IEEE 1516-2000 time and interval classes, as well as the IEEE 1516-2000 DLC time type can be overridden in LRC settings.

When a IEEE 1516-2000 federate joins, it uses the IEEE 1516-2010 time type of the federation to look up a time mapping. The mapping contains class names for the IEEE 1516-2000 time and interval factory. The IEEE 1516-2000 time and interval classes can be overridden in LRC settings. Note that the RTI actually ignores the time and interval classes provided in the *MobileFederateServices* parameter to the *joinFederationExecution* call.

When a Java IEEE 1516-2000 federate joins, it uses the IEEE 1516-2010 time type of the federation to look up a time mapping. The mapping contains class names for the Java IEEE 1516-2000 time and interval factory. The Java IEEE 1516-2000 time and interval classes can be overridden in LRC settings. Note that the RTI actually ignores the time and interval classes provided in the *MobileFederateServices* parameter to the *joinFederationExecution* call.

When a C++ IEEE 1516-2000 federate joins, it uses the IEEE 1516-2010 time type of the federation to look up a time mapping. The mapping contains class names for the Java IEEE 1516-2000 time factory and the IEEE 1516-2000 interval factory which are used by the LRC. The IEEE 1516-2000 time and interval classes can be overridden in LRC settings. The RTI uses the C++ time classes provided in the call to *joinFederationExecution*.

When a HLA 1.3 federate joins, it uses the IEEE 1516-2010 time type of the federation to look up a time mapping. The mapping contains class names for the HLA 1.3 time factory and the HLA 1.3 interval factory. The HLA 1.3 time and interval classes can be overridden in LRC settings. Note that the RTI actually ignores the time and interval classes provided in the *MobileFederateServices* parameter to the *joinFederationExecution* call.

When a C++ HLA 1.3 federate joins, it uses the IEEE 1516-2010 time type of the federation to look up a time mapping. The mapping contains class names for the Java HLA 1.3 time and interval factory. The HLA 1.3 time and interval classes can be overridden in LRC settings. The RTI uses the C++ time classes provided in the call to *joinFederationExecution*.

17.5 Data Distribution Management (DDM) Services

The Multi-API support in Pitch pRTI™ handles the differences in DDM between different APIs automatically to a certain extent. DDM in HLA 1.3 differs from the later standards in that it uses *routing spaces*.

Note that this is only a concern for federations using HLA DDM Services.

Federations created using HLA 1.3 FED

If a federation is created using HLA 1.3 FED file, the routing spaces and their respective *dimensions* are converted to IEEE 1516-2010 dimensions named as follows:

<routing space name>::<dimension name>

For example, if the HLA 1.3 FED defines a routing space named “RS1” with the dimensions “X” and “Y”, we will get the IEEE 1516-2010 dimensions named “RS1::X” and “RS1::Y”. The RTI will keep track of the HLA 1.3 routing spaces and dimensions and provide them to HLA 1.3 federates.

Federations created using a FOM (IEEE 1516-2010 or IEEE 1516-2000)

If a federation is created by an IEEE 1516-2010, IEEE 1516-2000 or IEEE 1516-2000 DLC federate which is using a FOM for creating, the routing space information will not be automatically available for HLA 1.3 federates. There simply is no such information in the FOM.

To deal with this, a mapping from IEEE 1516-2010/2000 dimension name to HLA 1.3 routing space and dimension have to be added to the LRC settings file.

It is added using the text editing tab, and each line for this mapping shall look like this:

```
hla13dimension_<IEEE 1516 dimension name> = <routing space name>::<HLA 1.3 dimension name>
```

For example, if the IEEE 1516-2010/2000 FOM has a dimension named "Dim1", we can map this dimension to HLA 1.3 routing space "RS" and dimension "X" with this entry in the LRC settings file:

```
hla13dimension_Dim21 =RS::X
```

17.6 Quick reference

The document ***prti_federate_quick_start.pdf***, which can be found in the *docs* subdirectory of the Pitch pRTI™ installation directory, contains details of how to setup PATH and CLASSPATH for each compiler type.

18 Common errors

This section lists common error messages along with solutions to each error.

18.1 Compiling C++ Federates

Problem: *I get the following error when I try to compile my federate in MS Visual Studio.*

```
libcpd.lib(xmbtowc.obj) : error LNK2001: unresolved external symbol __CrtDbgReport
Chat.exe : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.
```

Reason: You probably have not set the run-time library correctly in the project settings. See section 9.1.

Problem: *I get the following error when I try to compile my federate in MS Visual Studio:*

```
BaseFederateAmbassador.obj : error LNK2001: unresolved external symbol "public: virtual __thiscall
RTI::FederateAmbassador::~~FederateAmbassador(void)"
(??1FederateAmbassador@RTI@@@UAE@XZ)
```

Reason: Make sure that you have added the *librti1516e.lib* library file in the project settings. See section 9.1.

Problem: *When I try to compile my federate using MS Visual Studio, I get the following warning:*

```
warning C4541: 'dynamic_cast' used on polymorphic type 'class RTI::LogicalTime' with /GR-;
unpredictable behaviour may result
```

Reason: Make sure that you have enabled run-time type information for your project. See section 9.1 for more information.

18.2 Compiling Java federates

Problem: *When I try to compile my Java federate I get the following errors (among others):*

```
Chat.java:1: package hla.rti1516 does not exist
import hla.rti1516.*;
^
```

Reason: The compiler cannot find the *prti1516e.jar* file. Make sure you have the file available on your CLASSPATH. See section 9.5.2 for more information.

18.3 Starting Pitch pRTI™

Problem: *I try to start Pitch pRTI using the command prompt but I get the following error message:*

```
C:\Program Files\prti1516e>java se.pitch.prti1516e.RTIexec
Exception in thread "main" java.lang.NoClassDefFoundError: se/pitch/prti1516e/RTIexec
```

Reason: The file *prti1516e.jar* was not found by Java. Make sure the file is available on the CLASSPATH. See section 9.5.2 for more information.

Problem: *I try to start Pitch pRTI but I get the following message:*

```
RTIexec for pRTI(tm) 1516 v4.2 for IEEE 1516
Copyright (c) 2000-2010 Pitch Technologies AB, http://www.pitch.se

Enterprise Edition, licensed to Pitch Internal Development.
Could not start CRC listening on port 8989. Check if another instance of
pRTI is already executing or reconfigure the CRC port number.
Failed to initialize RTI.
Shutting down
```

Reason: Another instance of Pitch pRTI™ is already running on this machine. Try starting Pitch pRTI™ using a different port number if you need several instances running on the same machine. See section 16.1 for more information.

Problem: *When I try to start my federate I get the following error message:*

```
Incompatible version (909111111)
hla.rti1516e.RTIinternalError: Incompatible version (909111111)
```

Reason: The LRC and CRC you are using are different versions. Make sure that you use the same version for the CRC and all your LRC:s.

Problem: *When I try to start my federate I get the following message:*

```
Can't connect to RTIexec host. (909067002)
hla.rti1516e.RTIinternalError: Can't connect to RTIexec host. (909067002)
```

Reason: The federate could not connect to the RTI. This is probably because there is no CRC running on the specified IP address, or because the federate cannot connect to the specified address.

Problem: *When I try to start my federate I get the following error:*

```
Exception in thread "main" java.lang.NoClassDefFoundError:  
se/pitch/prti1516e/FederateAmbassadorImpl
```

Reason: The *jar* file containing the Pitch pRTI™ classes (*prti1516e.jar*) could not be found. Make sure that it is available on the CLASSPATH.

Problem: *When I try to start my C++ federate I get a dialog with the following text:*

```
This application has failed to start because rti1516e.dll was not found. Re-installing the application  
may fix the problem
```

Reason: The file *librti1516e.dll* could not be found. Make sure that you have this file on your PATH or that the file is located in the same directory as the executable file of your federate.

18.4 Running C++ Federates

Problem: *Starting a federate built with MS Visual C++ 8 or 9 fails with the error message "The application could not be initialized".*

Reason: MS .net framework is not installed and required by applications built with Visual C++ 8 or 9. For hosts that do not have MS Visual Studio installed, the MS .net framework redistributable package needs to be downloaded from Microsoft and installed.

Problem: *I'm using MS Visual Studio. My federate tries to send an interaction or receives an interaction from another federate. When either of these two events occur, a debug assertion fails with the following expression:*

```
_CrtIsValidHeapPointer(pUserData).
```

Reason: Make sure that you have set the correct run-time library in your project settings. See section **Error! Reference source not found.** for more information.

18.5 Federation Startup

Problem: *When I try to create the federation execution I get this error message:*

 null: line: 227 col: 29 expected: < found: eof004241E0

Reason: You are probably using a *FOM* file with the wrong extension or format. When running an IEEE 1516 federation the *FOM* file must be an *XML* file.

Problem: *I try to create the federation but I get the following error message:*

 Unable to open FDD file (909056001)

Reason: The *FOM* file specified in the create call was not found. Check the path and the name of the specified file.

Problem: *When I try to join a federation I get the following error message:*

 Federation execution does not exist (909029001)

Reason: The federation execution has not been created. Before you can join the federation it must be created. Make sure that you try to create the federation and check the CRC host address and port specified in the create call.

Problem: *When I try to join my federate I get the following message:*

 Unable to join federation, error when connecting to other joined federates (909056008)

Reason: When a federate joins a federation execution, it establishes a connection with every other currently joined federate. If this fails for some reason, the error message above is displayed. There can be several reasons why this occurs. Federates behind firewalls is one common cause. When using Linux, this error is fairly common because of the following networking issue:

When Pitch pRTI™ determines the IP address of the machine where a federate is running, it uses the Java function *InetAddress.getLocalHost()*. On Red Hat Linux installations, this function may return an *InetAddress* corresponding to the loopback address (127.0.0.1). This means that the federate informs the other federates that it is running at IP address 127.0.0.1, instead of the actual IP address of the machine. To fix this problem, edit the file */etc/hosts* in your Red Hat installation, and make sure that you enter the correct IP address of the host instead of 127.0.0.1.

Important Note: If you are running Windows XP, be aware that it contains a built in firewall. If you are experiencing this problem when running Windows XP check if the firewall is enabled. This can be done by first right-clicking on *My Network Places* in the *Start Menu* and selecting *Properties*. Then right-click on your network connection and select *Properties* again. Select the *Advanced* tab in the resulting window as shown in Figure 74.

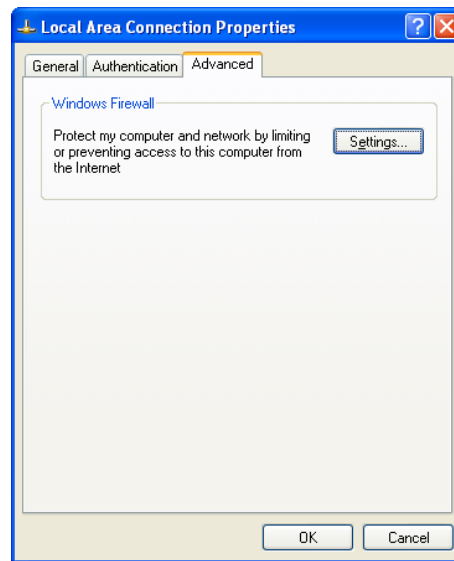


Figure 74 – Disabling Internet Connection Firewall on Windows XP.

To disable the firewall simply click the *Settings* button and turn off the firewall.

18.6 Get handles and register object instances

Problem: *When I try to get handles for my federate I get the following error messages:*

getInteractionClassHandle (909001001)

getObjectClassHandle (909001001)

Reason: This might indicate that the federate has not joined the federation execution correctly. Before any further operations can be performed the federate must join a federation execution. Make sure that you try to join a federation and check the name of the federation to join specified in the join call. Also check that the federation execution exists.

Problem: *I try to get handles for the object class Position, interaction class Trigger, attribute zcoord and parameter timeout but I get the following error messages:*

getObjectClassHandle Position (909149038)

getInteractionClassHandle Trigger (909149037)

Attribute zcoord not known. (909082002)

Parameter timeout not known. (909145002)

Reason: Position, Trigger, timeout and zcoord are not defined in the *FOM* file. All data used in the federation must be defined in the *FOM* file.

Problem: *I get the following error when I try to register an object and when I try to get an attribute handle of a certain object class:*

```
Object class cannot be null  
java.lang.NullPointerException: Object class cannot be null
```

Reason: You have probably not obtained the object class handle correctly. Make sure that you get the object class handle before you try to register any object instance or get any attributes of that class.

Problem: *I try to register an object but I get the following error message:*

```
RegisterObjectInstance: Object class not published (909087001)
```

Reason: Before you are allowed to register an object you must publish that object class. Make sure that the object class and attributes are published correctly.

Problem: *I try to register an object instance with a specified name but I get the following error:*

```
hla.rti1516.ObjectInstanceNameNotReserved: Fred (909000000)
```

Reason: The HLA 1516 standard requires that you reserve the object instance name before you register it. This is done with the *reserveObjectInstanceName* method, e.g.:

```
_rtiAmbassador.reserveObjectInstanceName(instanceName);
```

Note that you then must wait for the CRC to grant you reservation. The CRC will invoke the callback *objectInstanceNameReservationSucceeded* or *objectInstanceNameReservationFailed* depending on the result.

Problem: *I get the following error messages when I try to get a parameter or an attribute handle:*

Interaction Class Handle is null

Object Class Handle is null (909001001)

(909001001)

Reason: The interaction/object class handle that you send along with the get parameter/attribute handle call is probably not initialized. Make sure to get your interaction and object handles correctly before you get your parameter and attribute handles.

18.7 Updates and interactions

Problem: *I get one of the following errors when I try to update an object instance:*

Unknown object id 0 (909151008)004241E0

Object instance handle cannot be null

java.lang.NullPointerException: Object instance handle cannot be null

Reason: The object instance is probably not registered correctly. Make sure that the object class name is correct and that the registration is done before updating the instance attributes.

Problem: *I get the following error when I try to update and delete an object instance:*

Object Not Known, id = 115 (909147003)

Reason: The object instance does no longer exist in the federation. It has probably been deleted.

Problem: *When I try to update an object instance I get the following error message:*

UpdateAttributeValues: Attribute<name, 143, NotAcquiring 2 51> attribute not owned (909152021)

Reason: The attributes you try to update are not owned by your federate. Make sure that it is the right object attributes you are updating. If the object instance is not registered by your federate you must request ownership of its attributes before you are allowed to update them.

Problem: *When I update attributes I get the following error message:*

Connection reset by peer: socket write error
java.net.NoRouteToHostException: No route to host: Datagram send failed

Reason: The local LRC is unable to connect to the remote LRC. Typically, the remote LRC cannot be reached because of an intervening firewall, or because an intermediate router is down.

18.8 Time management

Problem: *I'm using time management in my federate and enableTimeConstrained() works fine, but after that I cannot receive any interactions or updates.*

Reason: If your federate is time constrained you must call `_rtiAmbassador->enableAsynchronousDelivery()` in order to get receive order interactions and updates at any time.

Problem: *When I try to advance time in my federate I get the following exception:*

Time Advance Already InProgress (909128003)004262F8

Reason: Your federate has already tried to advance time and has still not obtained a grant for that request. You have to wait for a grant before you can make a new request.

Problem: *When I try to update an attribute with a time stamp I get the following error message:*

Invalid federation time (909152011)004262F8

Reason: The time stamp you have given in the update call is not valid. It is probably lower than some federate's logical time.

18.9 Miscellaneous

Problem: *The federation performance is very low. Earlier the federation has run much faster!*

Reason: Check if the tracing is activated. The tracing is very useful but might also reduce the federation performance significantly.

Problem: *Sometimes my callback seems to get lost. My code looks like this:*

```
1 _rtiAmbassador.reserveObjectInstanceName(name);
2 synchronized (_reservationSemaphore) {
3     try {
4         _reservationSemaphore.wait();
5     } catch (InterruptedException e) {
6     }
7 }
8 System.out.println("Reservation completed");
```

The callback looks like this:

```
void objectInstanceNameReservationSucceeded(String name) {  
    synchronized (_reservationSemaphore) {  
        _reservationSemaphore.notify();  
    }  
}
```

From time to time, the program gets stuck at the wait() method and never reaches the print statement.

Reason: This is a common mistake in multi-threaded programming. What happens is that the callback is delivered before line 2 is executed, i.e. before the caller has synchronized the semaphore. Instead, the callback synchronizes the semaphore and calls notify. After the callback has terminated, the caller continues at line 3 where it synchronizes the semaphore and calls wait. This call will never terminate since the callback has already happened.

Solution: Make sure that the caller synchronizes the semaphore before calling the RTI.

```
1 synchronized (_reservationSemaphore) {  
2     try {  
3         _rtiAmbassador.reserveObjectInstanceName(name);  
4         _reservationSemaphore.wait();  
5     } catch (InterruptedException e) {  
6     }  
7 }  
8 System.out.println("Reservation completed");
```

Problem: *I am experiencing packet loss, missing callbacks and instable behavior on a computer with local firewall software.*

Reason: Unless you explicitly configured the firewall and the RTI to allow traffic on certain ports, you should not put a firewall in between or on computers in a federation. Disable or remove the firewall to verify that this is indeed the problem. See chapter 14 on how to configure the Pitch pRTI™ when operating with firewalls.