Pitch pRTI™ User's Guide

«Make Your Systems Work Together»

Version 5.5

Table of Contents

1. Introduction

1.1. About This Document

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 set up 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 is a leading supplier of commercial, standards-based infrastructure tools and services to the modeling and simulation community.

Well-known tools from Pitch include:

Pitch Booster™

Connect simulations across WAN with performance and fault tolerance.

Pitch CDS Gateway™

Connect federations of different security classifications.

Pitch Commander™

Monitor and control your simulations across your organization.

Pitch Debrief™

Add synchronized debrief capabilities to simulation and training exercises

Pitch Developer Studio™

Generate a royalty-free HLA and DIS module in C++, C# or Java for your simulation.

Pitch DIS Adapter™

Integrate your DIS applications and your HLA federations with monitoring and filtering.

Pitch Extender™

Connect federations with different RTIs, FOMs or different performance profiles.

Pitch KML Adapter™

Visualize your scenario in the KML viewers such as Google Earth.

Pitch Media Streamer™

Integrate video and simulation data in your simulation infrastructure.

Pitch pRTI™

Make your systems work together using the open international HLA standard.

Pitch Recorder™

Record, inspect, play back and export your HLA, DIS, Link 16 and other simulation data.

Pitch Talk™

Communicate using voice, intercom, radio and chat with scalability and management.

Pitch Unreal Engine Connector™

Connect your Simulation to the Unreal Engine using HLA.

Pitch Visual OMT™

Create, maintain, visualize and analyze object models for information exchange.

Read more about Pitch Technologies at www.pitchtechnologies.com.

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 RTIs 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 website www.pitchtechnologies.com.

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 which 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.pitchtechnologies.com 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 15.6.
- **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 www.pitchtechnologies.com 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 APIs 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.

Pitch pRTI™ includes HLA 4 Preview features. This gives users the opportunity to experiment with some of the new features being developed for the next version of the HLA standard whose working name is HLA 4. The HLA 4 Preview features are subject to change. The following HLA 4 Preview features are available.

Federation Modes

pRTI provides two different federation modes: Evolved Mode and HLA 4 Mode.

Evolved Mode

The Evolved Mode provides perfect backwards compatibility for Evolved federates. An Evolved Mode federation has the following properties, restrictions and requirements:

- A federation will be set in Evolved Mode if it is created by an Evolved federate using an

Evolved FOM, even if the enableHLA4preview flag is set.

- HLA Evolved federates without enableHLA4preview flag set will be accepted. This includes federates running on earlier LRC versions.
- HLA Evolved federates will observe perfect Evolved behavior. For example, dimensions will use Evolved attribute-based dimensions rather than HLA 4 object class-based dimensions.
- HLA 4 federates may **not** join an Evolved Mode federation. This is to ensure that Evolved Mode federations maintain maximum backwards compatibility.
- HLA 4 FOM modules are not allowed. An attempt to load HLA 4 FOM modules into the federation will be rejected with an exception.

HLA 4 Mode

An HLA 4 Mode federation provides HLA 4 behavior for HLA 4 federates while allowing Evolved federates to partake in the federation. An HLA 4 Mode federation has the following properties, restrictions and requirements:

- A federation will be set in HLA 4 Mode if it is created by an HLA 4 federate.
- A federation will also be set in HLA 4 Mode if it is created by an Evolved federate using an HLA 4 FOM and the enableHLA4preview flag is set.
- HLA 4 federates will observe HLA 4 behavior. For example, dimensions will use HLA 4 object class-based dimensions rather than Evolved attribute-based dimensions.
- HLA Evolved federates may join an HLA 4 Mode federation if the enableHLA4preview flag is set. The HLA Evolved federates may observe behavior that is not fully HLA Evolved compliant.
- HLA Evolved federates without enableHLA4preview flag set will be rejected. This includes federates running on earlier LRC versions.
- HLA Evolved FOM modules are allowed. HLA Evolved FOM modules will be internally converted to HLA 4. During conversion, attributes-based dimensions will be converted to object class-based dimensions.

HLA 4 Java API

HLA 4 adds new functionality which means that the API is updated. Pitch pRTI™ supports the new HLA 4 Java API. This is a draft API that will change. Note that some of the new HLA 4 services may not do anything since Pitch pRTI™ does not support them yet.

HLA 4 C++ API

HLA 4 adds new functionality which means that the API is updated. Pitch pRTI™ supports the new HLA 4 C++ API. This is a draft API that will change. Note that some of the new HLA 4 services may not do anything since Pitch pRTI™ does not support them yet.

Extended merge

Extended merge makes it possible when merging modules to:

- Add more attributes to existing object classes.
- Add more parameters to existing interaction classes.
- Add more enumerators to existing Enumerated datatypes.
- Add more alternatives to existing Variant Records.
- Add Dimensions to existing attributes and parameters.

Note that these extended merging operations only are available when the federation is created. Extended merging is not allowed as part of the join operation. For more information on extended merging, see https://pitchtechnologies.com/2019/01/new-object-modeling-opportunities-in-hla-4/

Strict vs Relaxed Advisories

HLA 4 introduces an alternative way to calculate advisories that is more scalable. In HLA Evolved (IEEE 1516-2010), advisories are calculated based on the known class of instances. Distributing this information about every instance in the federation requires a lot of network bandwidth and CPU load, especially when federations grow large.

In HLA 4, *Strict Advisories* are calculated in the same way as in HLA Evolved. In the new, more scalable variant, *Relaxed Advisories* are calculated based on subscription information. This means that a lot less information needs to be distributed by the RTI.

The difference in the advisories delivered to federates are minor and restricted to a few corner cases where the known class doesn't match the current subscription. The typical case where this can happen is if a federate first subscribes to the attributes *x*, and *y* on *superclass*, discovers an instance as *superclass*, and then the federate subscribes to the attributes *x*, *y*, and *z* on *subclass*. The federate's known class for this instance is now *superclass*, and it will only receive updates to the attributes *x* and *y*. With *Strict Advisories*, the known class is taken into account and the owning federate will only be advised about attributes *x*, and *y*. With *Relaxed Advisories*, the advisory will be based on the subscription alone and the owning federate will be advised about attributes *x*, *y*, and *z*.

To avoid these corner cases, the general rule is to always subscribe to subclasses before superclasses.

Authorized Federate

With HLA 4, it is possible to restrict access to federation executions via an Authorization Service, which may be provided by third parties.

Based on credentials, provided by each federate in the call to `connect`, an Authorization

Service may disallow calls to connect to the RTI, create and destroy a federation execution, or join a federation execution.

The HLA 4 standard mandates that a built-in Authorization Service, HLAauthorizer, be provided by every RTI. This Authorization Service can restrict access to federation executions based on shared plain-text passwords. The detailed rules of the service is left up to the individual RTI. More advanced Authorization Services may be developed by third parties according to the HLA 4 standard API, and configured to be used by the RTI at runtime.

To control the settings for the Authorized Federate feature in Pitch pRTI™, use the CRC settings tool. The *Authorization* tab contains settings for this feature.

Any third-party authorization service you wish to use must be implemented according to the HLA 4 standard, and added to the class path of Pitch pRTI™. To use the third-party authorization service, on the *Authorization* tab, select *Use custom HLA authorizer* and enter the service name of the custom authorizer. If the named service can not be found at runtime, Pitch pRTI™ will warn about this and refuse to start.

Once an authorization service is active, it applies to all federates (both HLA 4 and Evolved), and all their calls to `connect`, `createFederationExecution`, `destroyFederationExecution`, and `joinFederationExecution`.

HLAauthorizer is the built-in authorization service, which is the default value for the setting. This is a very simple service, which allows full access to federations based on plain-text passwords.

You can set passwords to individual federations by name, as well as a general password. Federations that do not have any specific password will use the general password.

A federate can connect to the RTI if its credentials matches the general password or any of the federation-specific passwords.

A federation with a specific password can be created, destroyed, and joined by a federate with credentials matching the specific password.

A federation without a specific password can be created, destroyed, and joined by a federate with credentials matching the general password.

If no general password is set, then no password is required to connect to the RTI and to create, destroy, and join federations that have no specific password.

If no password is set, and the federate provides a password, then the federate will be rejected. This is based on the principle that a federate that provides a password expects a secure environment.

The HLA 4 API `connect` method accepts a `Credentials` argument that is used with the

Authorized Federate feature.

The provided credentials can be overridden with a plain-text password by using the LRC Settings tool. Find the settings for this on the *Overrides* tab. These settings accept a plain-text password for use with the default HLAauthorizer. When the override is enabled, the configured password will be provided to the RTI in place of the Credentials passed to `connect`.

A federate using the Evolved HLA API can also use the Authorized Federate functionality by enabling the password override in LRC Settings.

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 below.

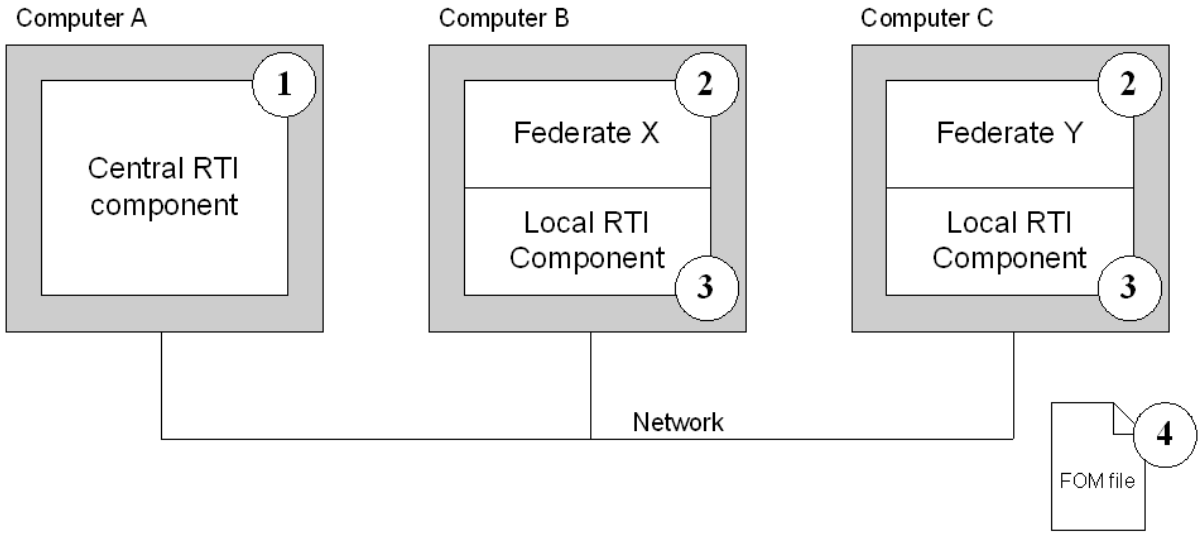

Figure 1. An example topology.

The environment in the figure above 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 is that shown in the figure below.

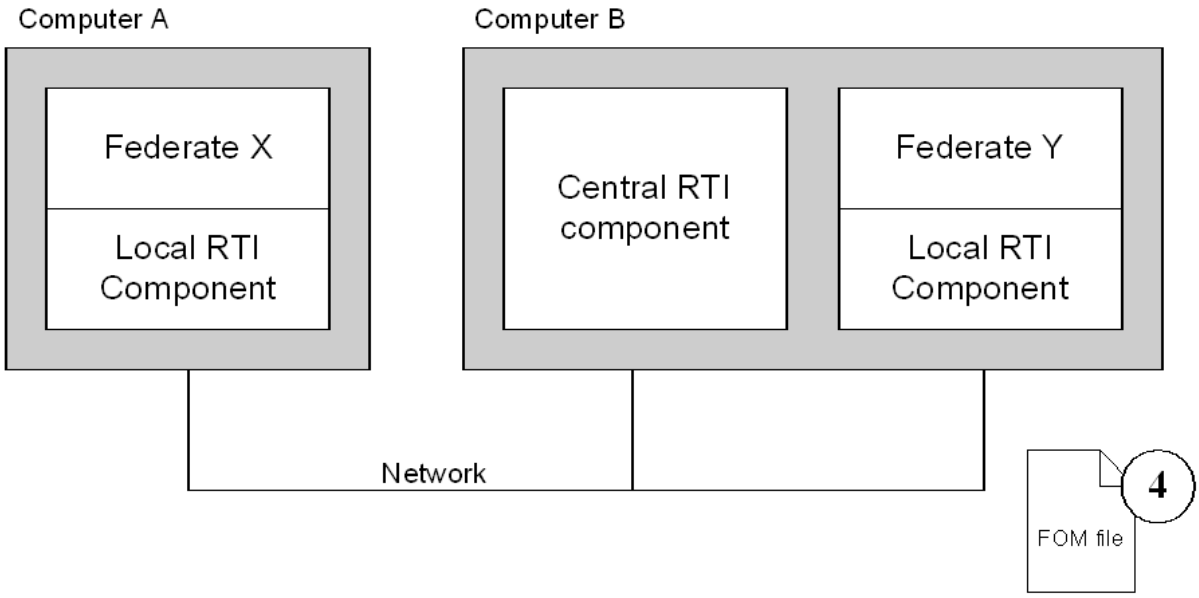

Figure 2. Another example topology.

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 LRCs (one for each federate). The figure below illustrates the relationship between the CRC, the LRC and the federates.

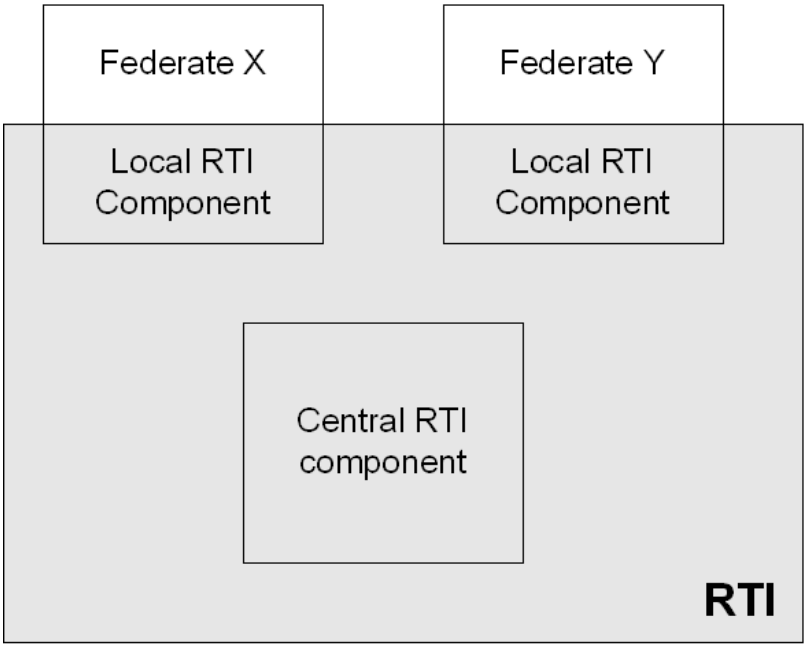

Figure 3. The relationship between the CRC, the LRC and the federates.

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. The figure below shows a screenshot of the tool.

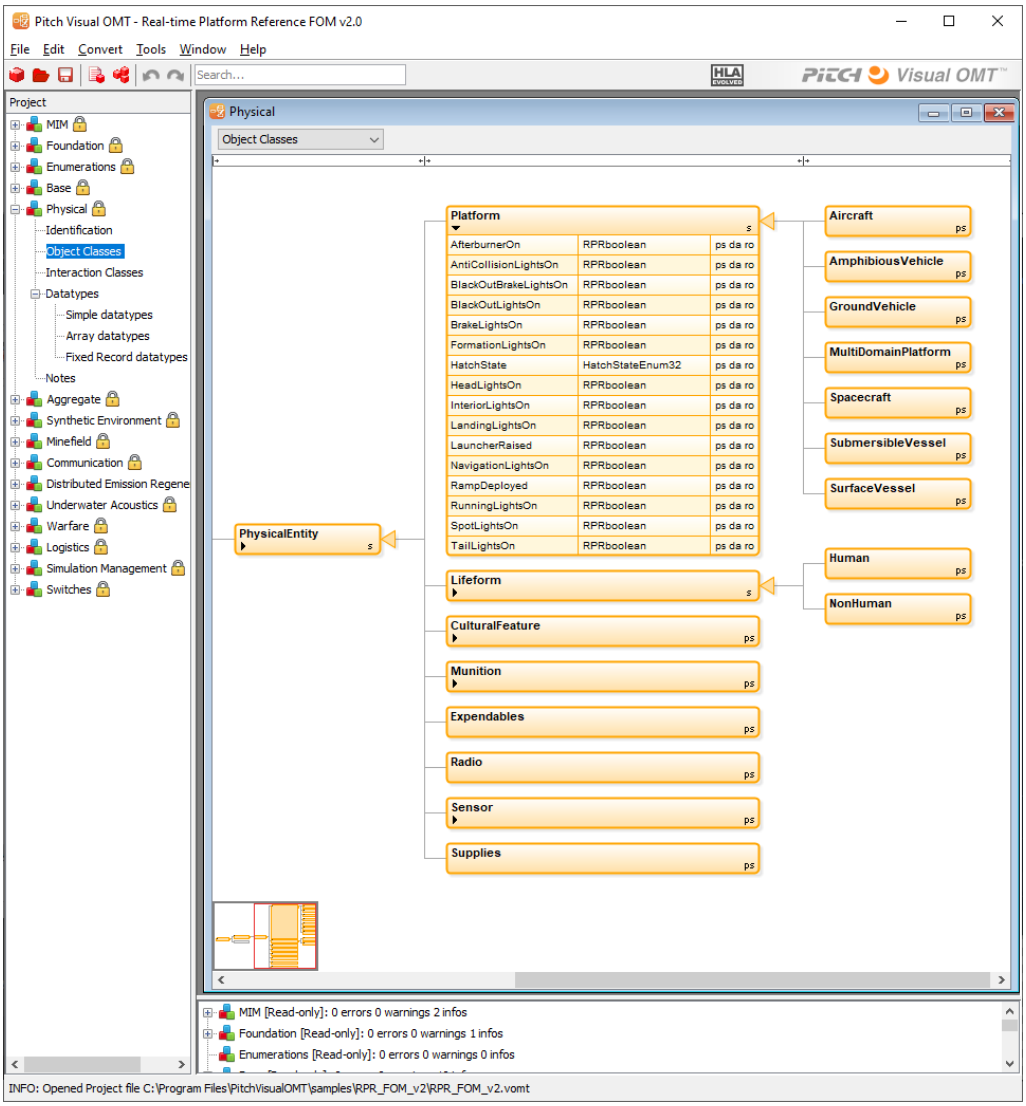

Figure 4. Pitch Visual OMT™ object modeling tool.

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.

You may mix different brands and operating systems in the same federation.

2.7. The Network

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.

- Wi-Fi. All computers are equipped with a Wi-Fi card and establish wireless connections through a Wi-Fi 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 into 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 WANs with routers, for example over the Internet or big corporate networks. See Section 15 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 the figure below.

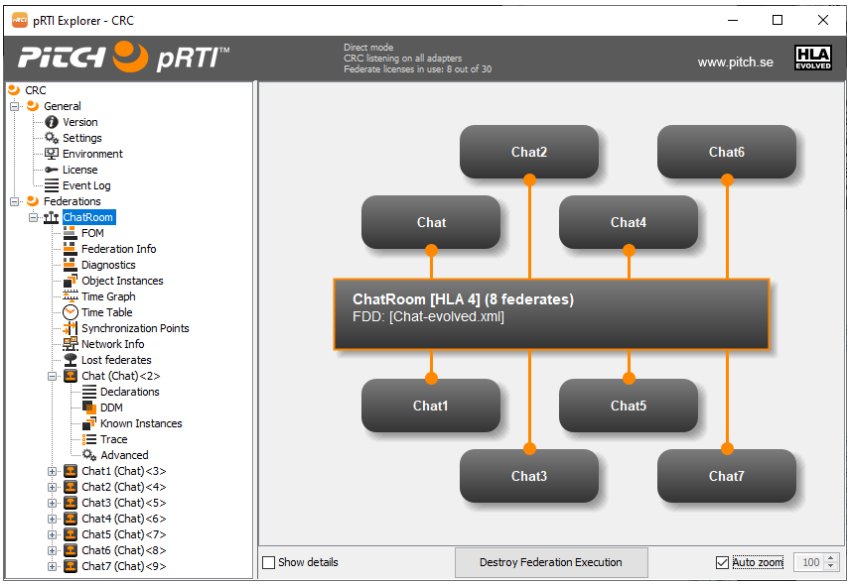

Figure 5. Desktop CRC GUI.

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.

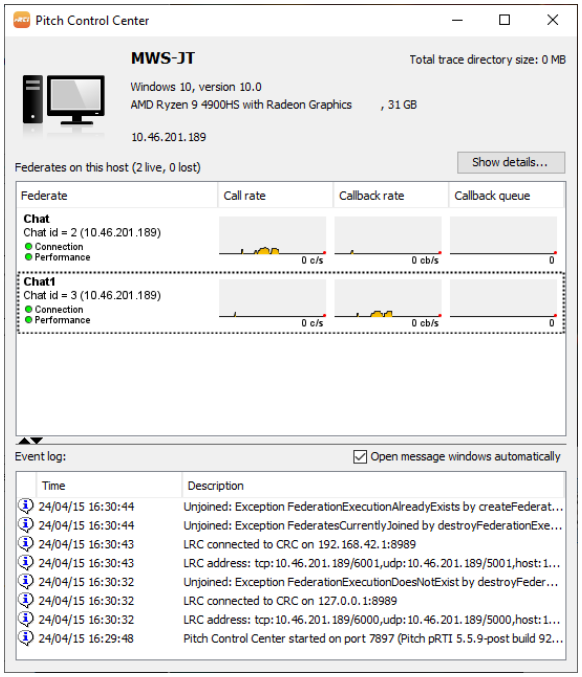

Figure 6. Pitch Control Center.

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.

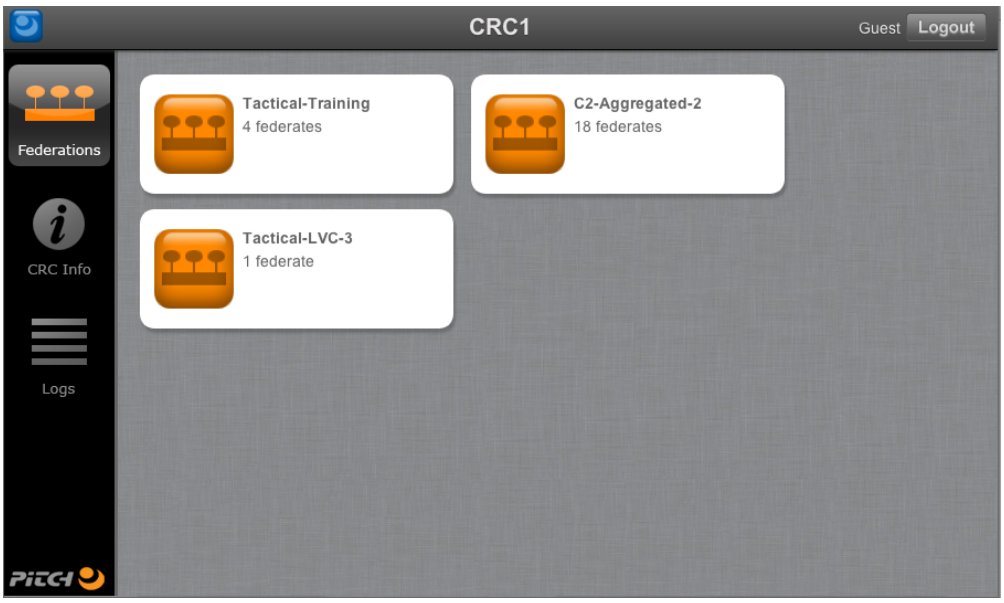

Figure 7. Pitch Web View.

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.

If upgrading an existing installation that is running as a service, please inform users of the maintenance and stop the service before upgrading the software.

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

The figure below shows the introduction screen which gives you a general introduction to the installation.

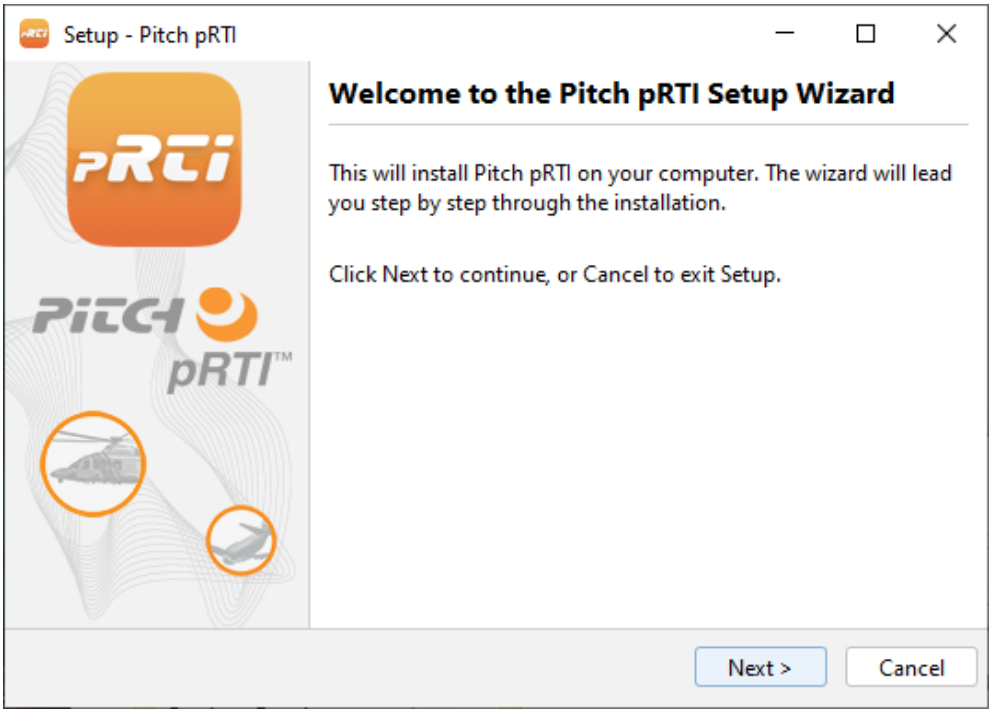

Figure 8. Installation introduction.

Click **[Next]** to continue. The license agreement will then be presented to you as shown in the figure below.

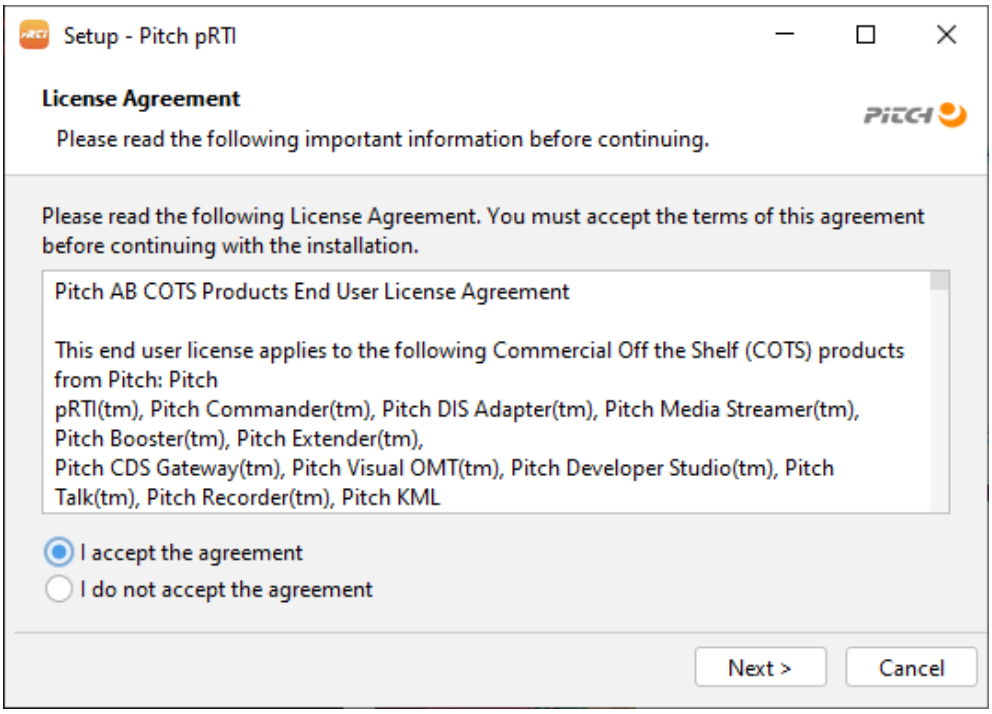

Figure 9. End user license agreement.

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™

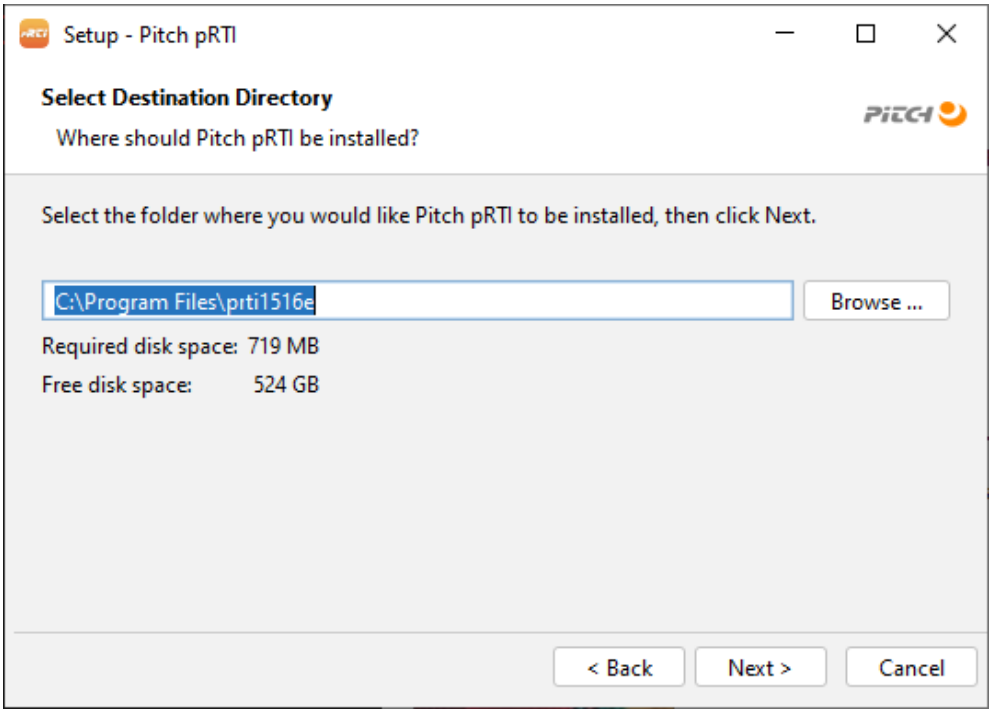

Figure 10. Installation destination.

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

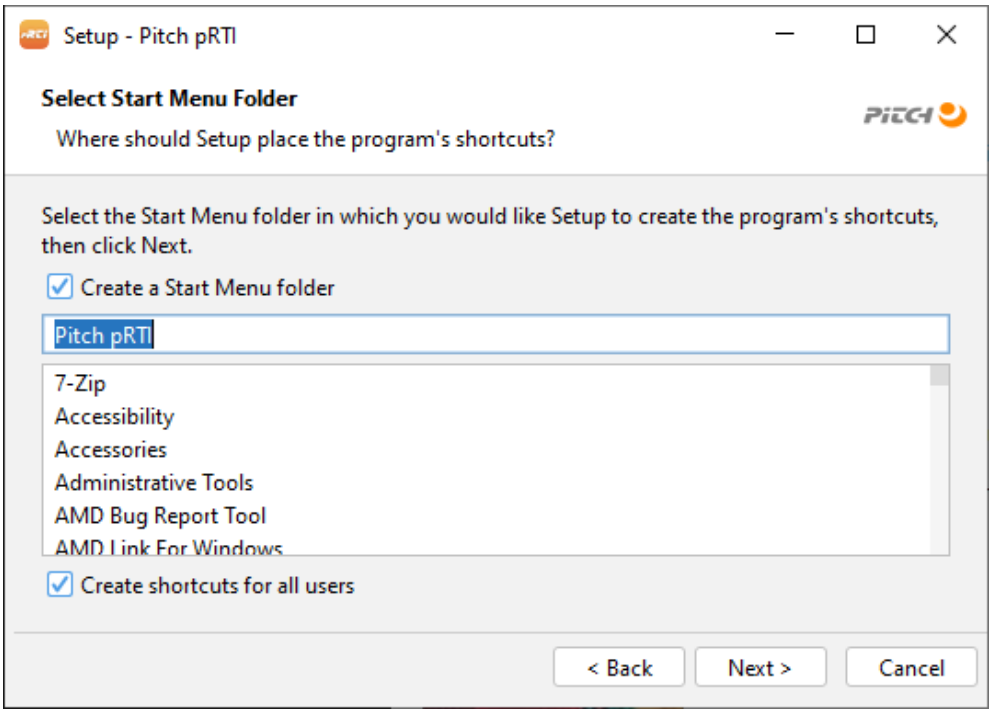

Figure 11. Create shortcuts on the start menu.

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.

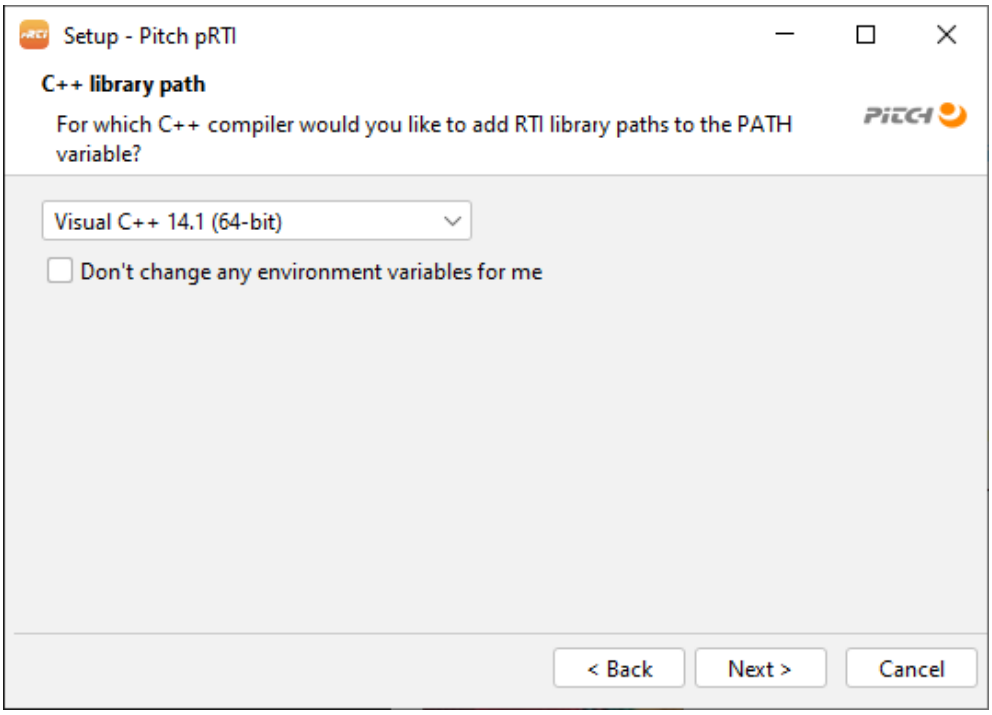

Figure 12. C++ library PATH setting.

Click **[Next]** to continue. You are then presented additional options such as creating desktop icons for the pRTI™ CRC, installing the Web View Server (Pitch pRTI™'s web interface), and opening ports in the firewall.

pRTI's CRC has to be reachable by federates in your network. By default, Windows does not allow inbound connections to the CRC's network ports. Checking the option "Open required ports in Windows Firewall" adds the necessary rules to the firewall, making the CRC reachable by other hosts.

If you have special needs for your network, uncheck this option in the installer and add the firewall rules you require manually. See Section 15.2 and Section 15.4 for further details.

Important note: The option to open the required ports at the firewall is only available in the Windows installer. If you are installing pRTI™ on Linux or macOS you may need to do this manually if you observe that federates cannot reach your CRC.

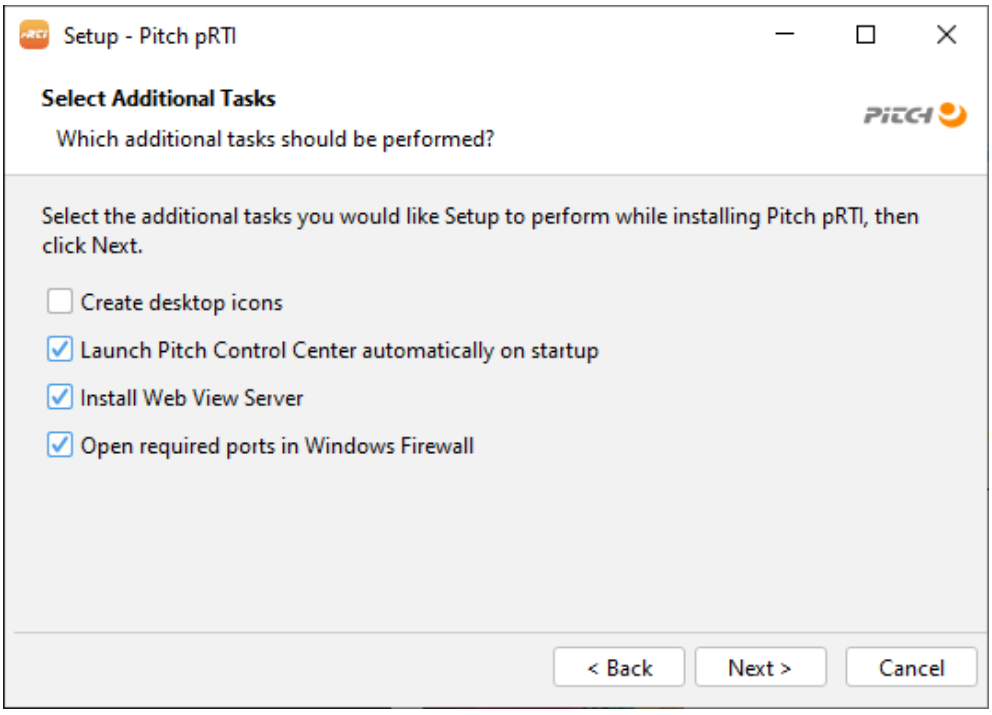

Figure 13. Additional options.

Click **[Next]** to continue. You can now choose a name to identify this CRC in the network.

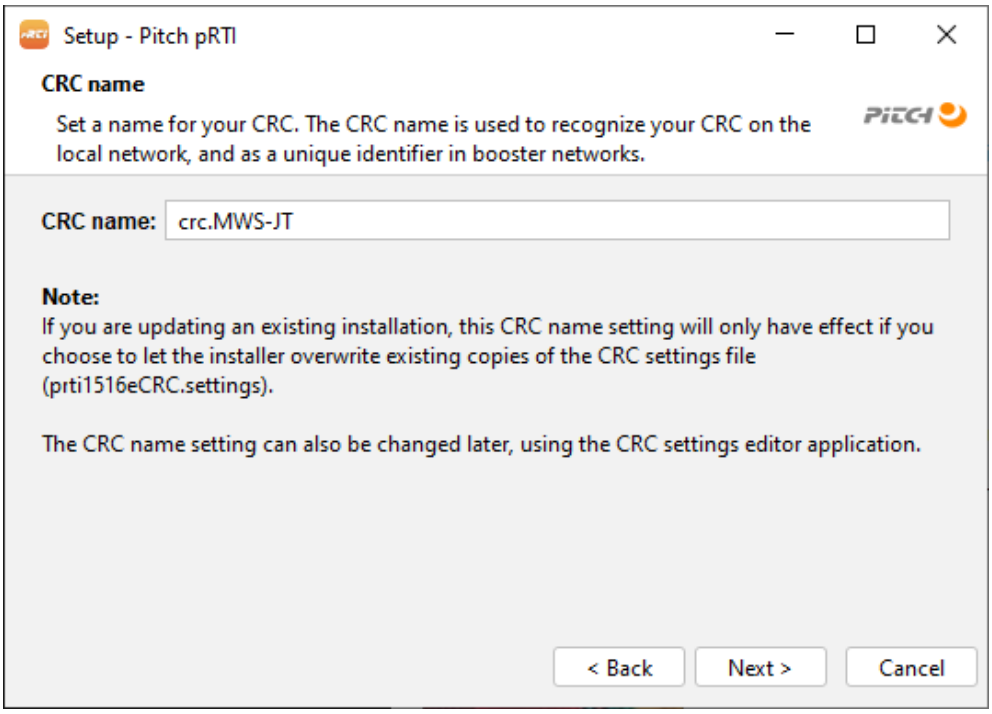

Figure 14. Set CRC name.

Click **[Next]** to finish the installation.

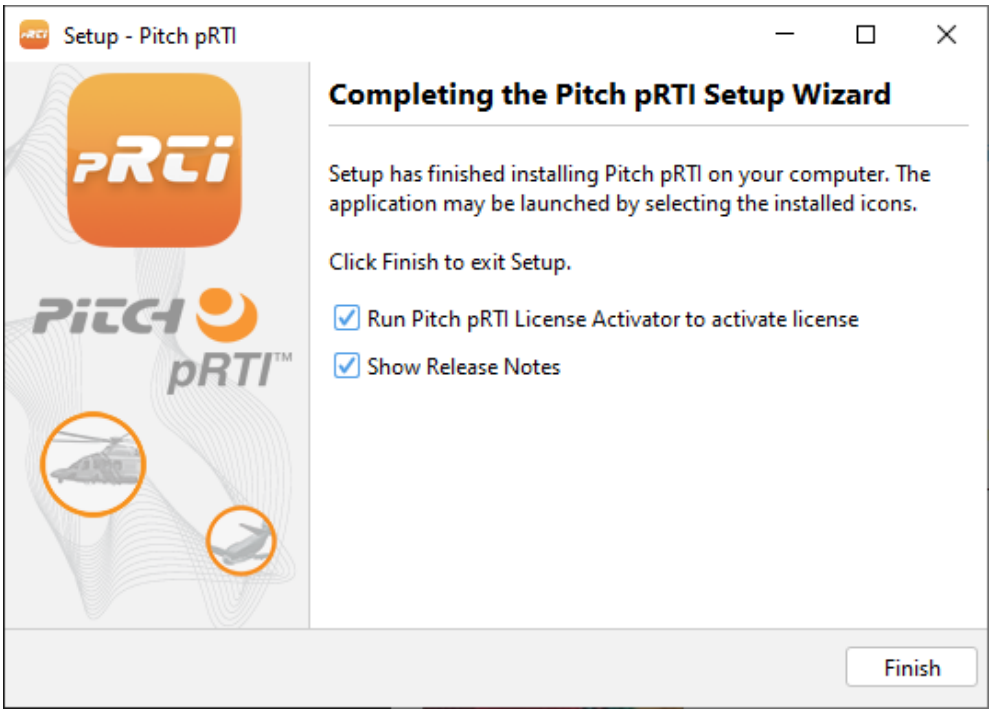

Figure 15. Completing the installation.

Click **[Finish]** to exit the installer.

3.2. Command line options for the installer executable

The installer executable optionally can be launched with command line options. The subset of options which could be useful are described in the table below.

Option	Description
-h or -help or /?	Displays the complete list of options
-c	Executes the installer in console mode
-q	Executes the installer in unattended mode
-console	If the installer is executed in unattended installation mode and -console is passed as a second parameter, status messages will be printed on the console from which the installer was invoked.
-overwrite	Only valid if -q is set. If -overwrite is set, the installer will overwrite files where an installer in attended mode would have asked the user.
-wait [timeout seconds]	If -wait is passed in unattended mode, the installer will avoid locking problems and wait for other installers to finish.
-dir [directory]	Sets an alternative installation directory in unattended installation mode.
-Vinstall-firewall-rules=[**true** | false]	Opens port 1099, 8989 and 55555 for pRTI™ and pRTI™ Service. Also opens required ports for the Federate Protocol Server. Only available on Windows.

3.3. 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 the License Activator app needs to be executed with writing permissions to your pRTI installation directory.

3.3.1. 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.

3.3.2. 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.

The CRC settings file can also be edited with a text editor, for example if the graphical CRC Settings tool isn't available. Add/replace the following entries to the CRC settings file.

```
CRC.licenseType=server
CRC.license-server.host=<my-license-server>
CRC.license-server.federateCount=<count>
```

3.4. 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 will start 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)

Select the License view in the navigation tree under General. If there is no license showing

you will need to run the License Activator tool first.

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.

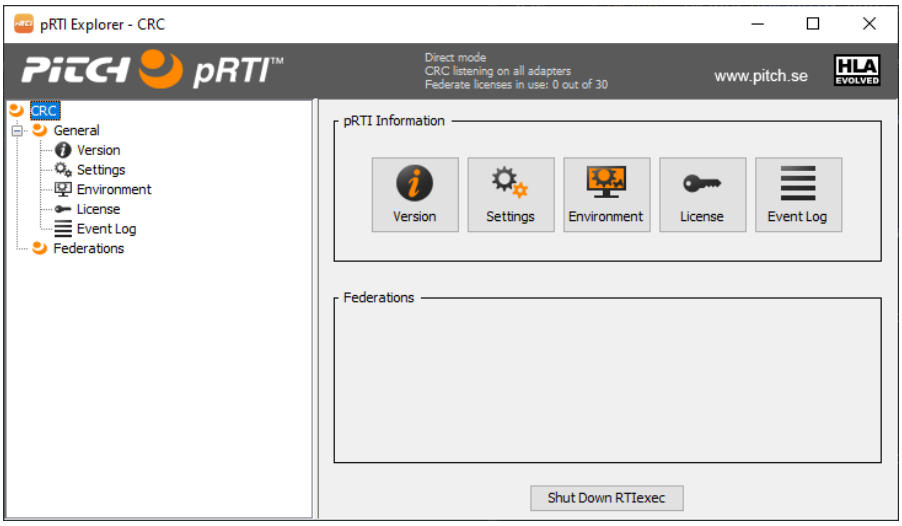

Figure 16. The Pitch pRTI™ Explorer.

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™.

3.5. 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.

3.6. 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 17.2 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 the figure below.

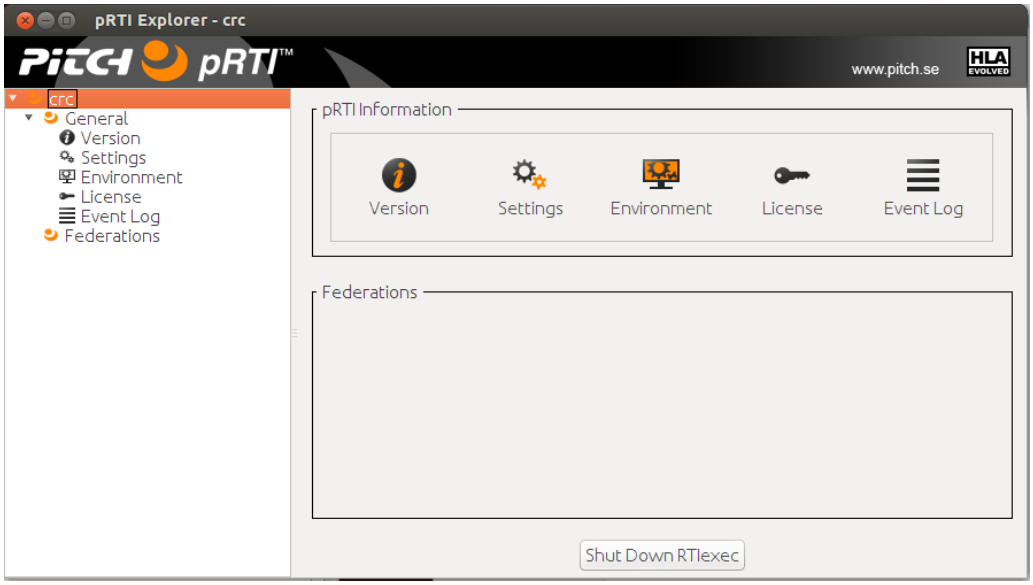

Figure 17. The pRTI™ Explorer

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-1516e/
[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 a C++ Chat federate, open another command prompt window and run for example the file named `chat-cpp-1516e_gcc73_64.sh`.

```
[root@Enorm root]# cd /opt/prti1516e/samples/chat-cpp-1516e/
[root@Enorm bin]# ./chat-cpp-1516e_gcc73_64.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 `Enter` 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 `Enter` 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 `Enter` key twice.

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

3.7. 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 uninstallation 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 the uninstallation 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 Section 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 9. 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 the figure below will be displayed.

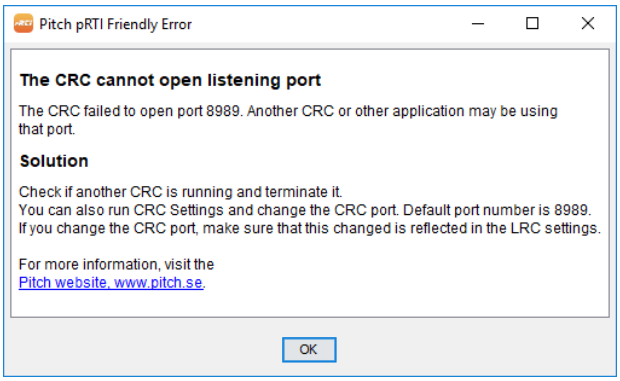

Figure 18. Error message when multiple CRC instances are started.

Either shut down the other CRC or specify a different port as described in Section 17.1. 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.

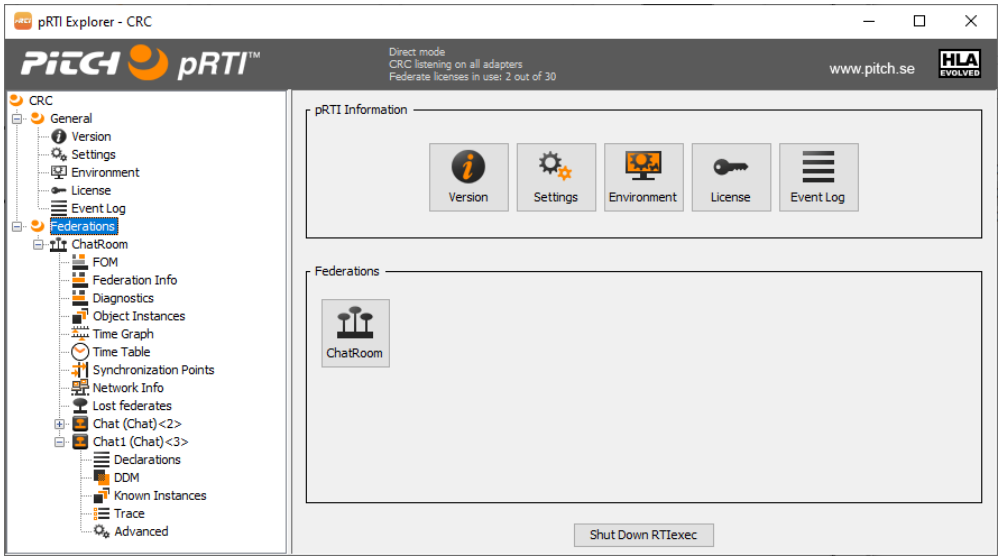

Figure 19. Pitch pRTI™ start panel.

The figure above 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

The figure below 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.

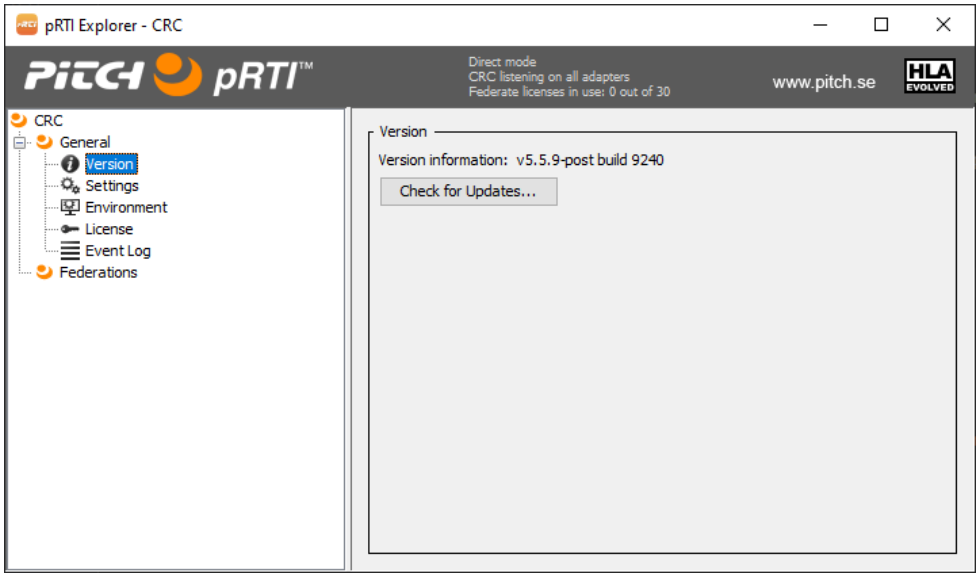

Figure 20. The version panel.

5.2.2. Settings

The figure below shows the settings panel. For more information on this see Section 17.1.

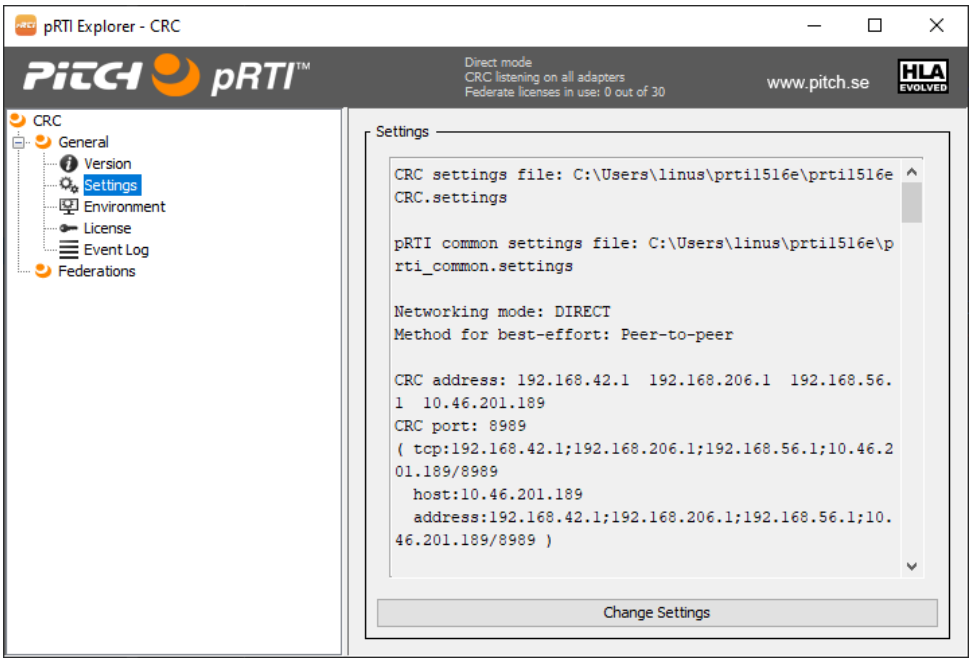

Figure 21. The settings panel.

5.2.3. Environment

The figure below shows the environment panel. This provides information on which JRE is being used and other environmental data.

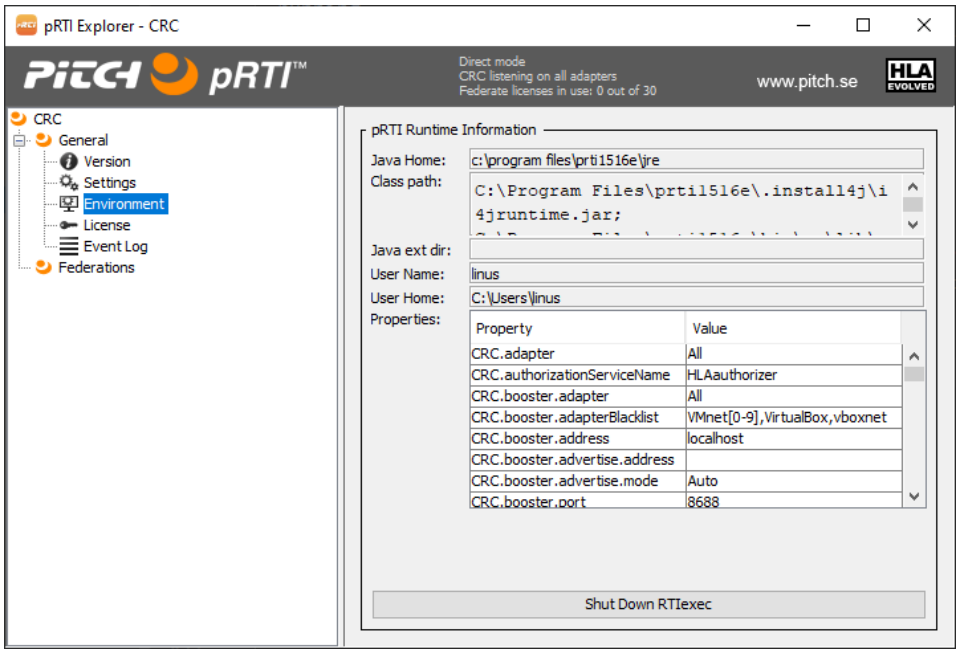

Figure 22. The environment panel.

5.2.4. License

The figure below 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.

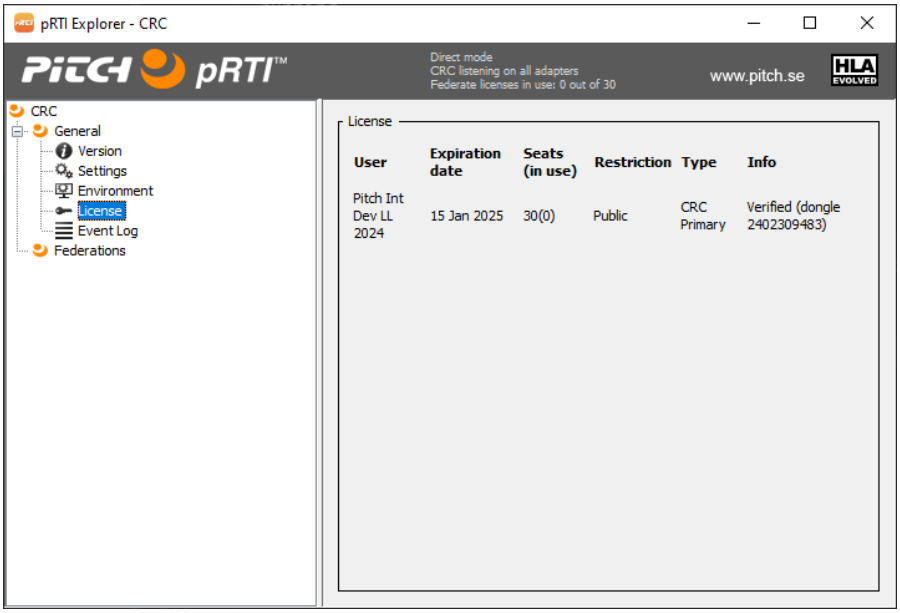

Figure 23. The license panel.

5.2.5. Event Log

The figure below 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. Double clicking av event in the event log opens a dialog box with the event details.

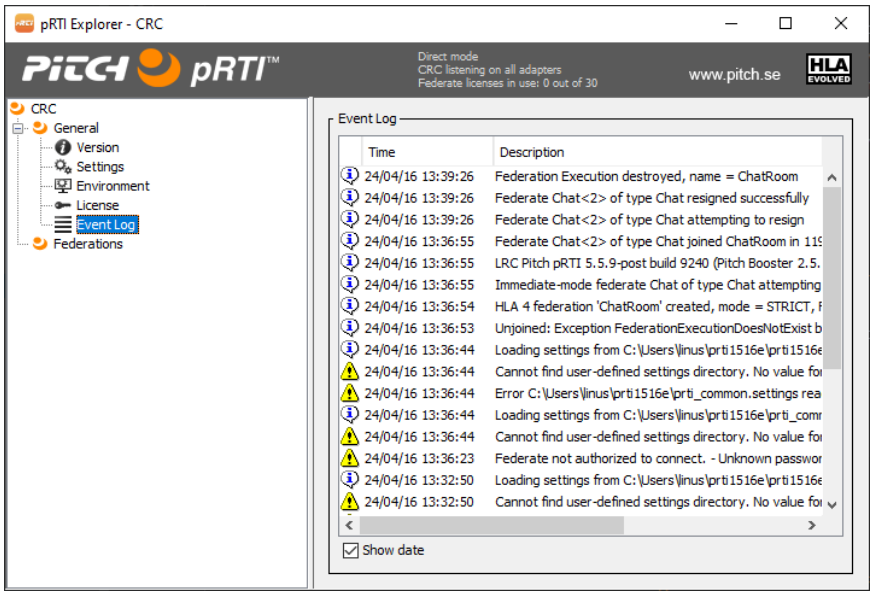

Figure 24. The event log panel.

5.2.6. Federation Overview

Clicking on the *ChatRoom* federation icon brings you to the view shown in the figure

below. This shows information about the federation, including information about the Central RTI Component and each participating federate.

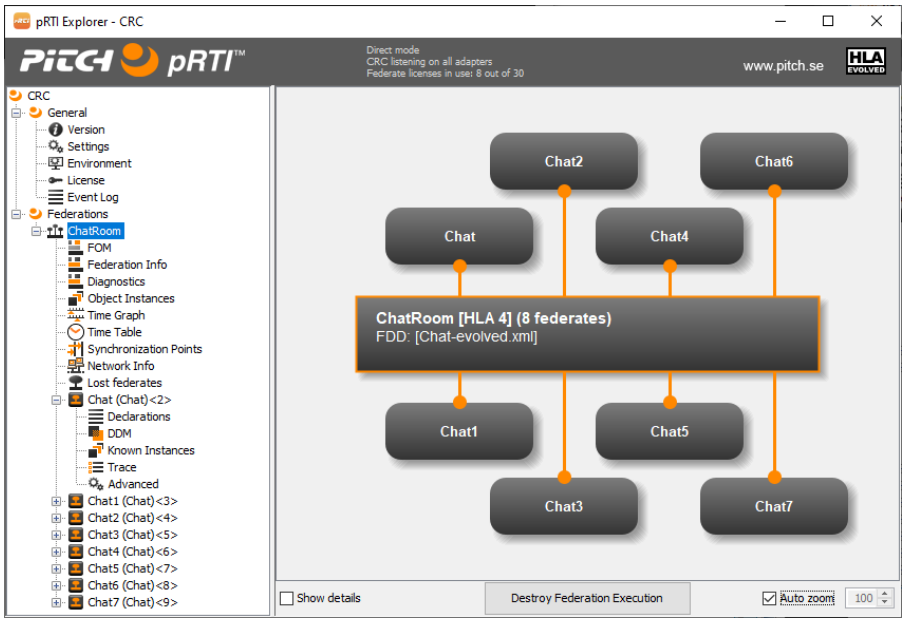

Figure 25. The federation lollipop view.

5.2.7. Federation Object Model (FOM)

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

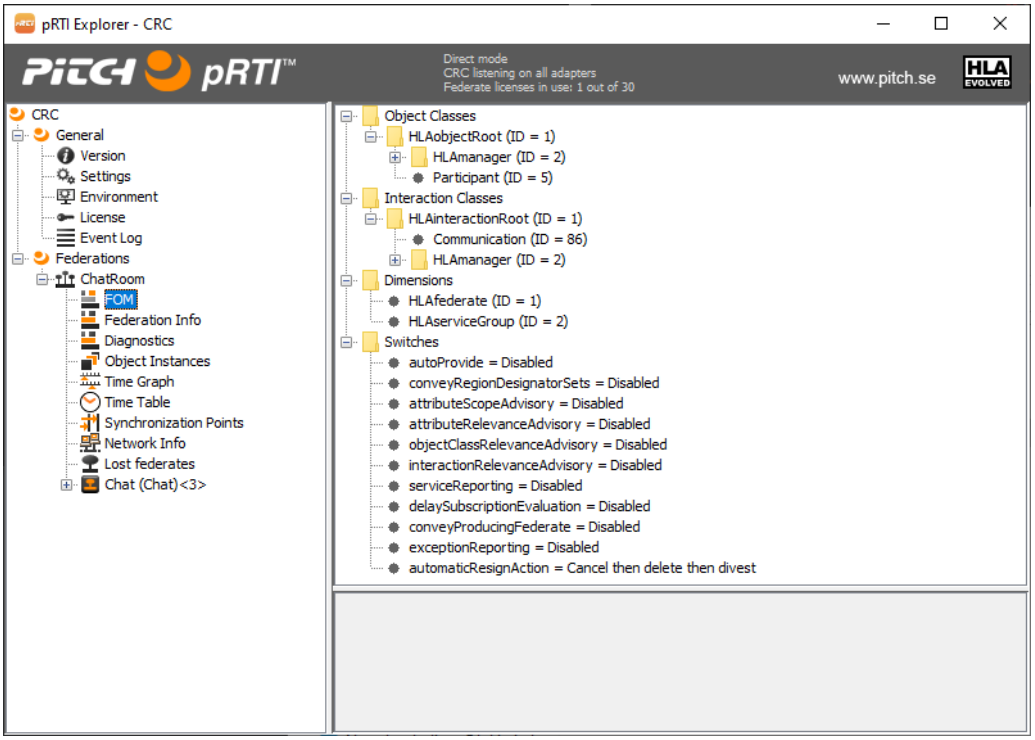

Figure 26. Inspection of the FOM.

5.2.8. Federation Info

The Federation Info view lets you inspect the FOM modules that were used in the selected federation at runtime. The figure below illustrates how the Federation Info view displays the modules that are in use by the federation. It also shows whether HLA 4 Extended merging was used.

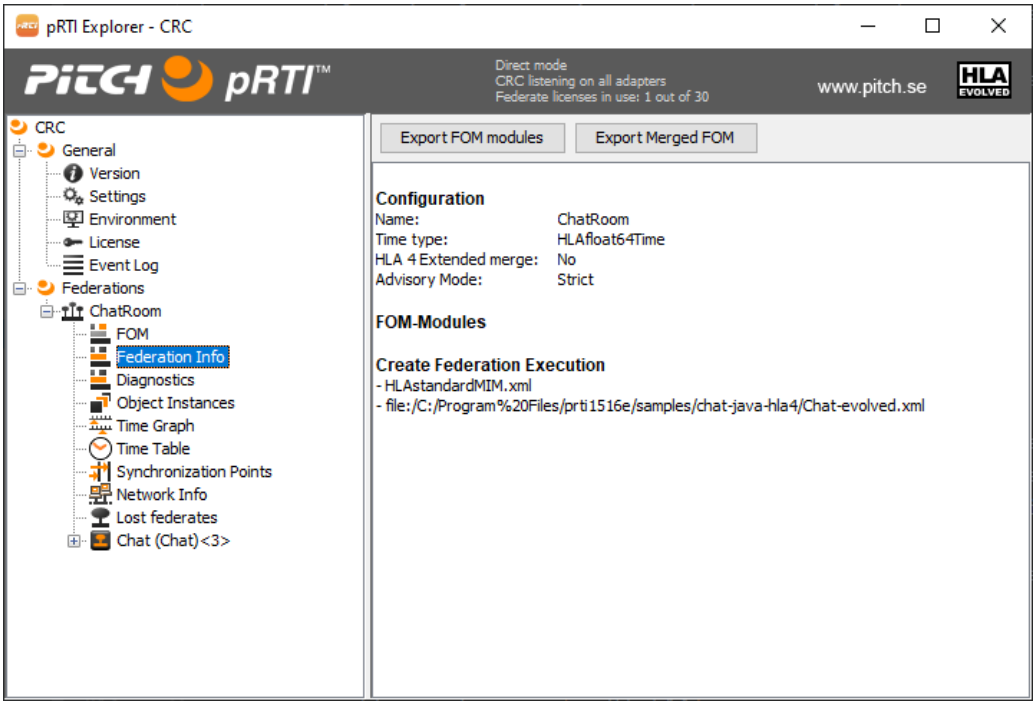

Figure 27. Inspection of the FOM modules.

5.2.9. Diagnostics

The Diagnostics view lets you perform diagnostics in the unlikely case that a federate gets stuck while trying to join. The *Diagnose join* button generates a short report in the Diagnostics panel. The *Diagnose join (full)* button generates an extended report that is displayed in the Diagnostics panel, but also stored as an html file which is opened in a web browser. An entry for this file is added in the Event log where it can be accessed at a later time.

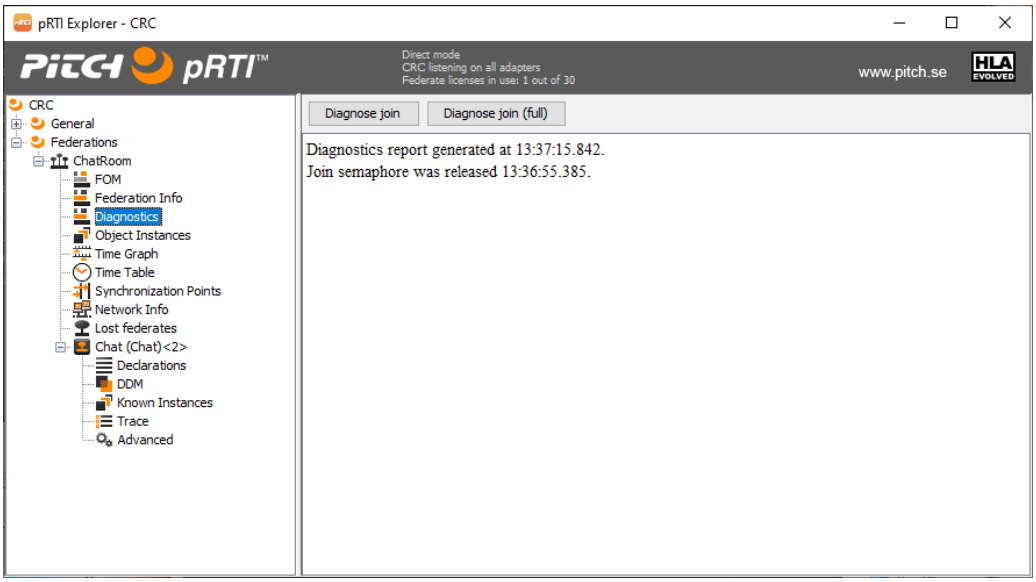

Figure 28. Diagnostics.

To access a stored report, find the corresponding entry in the Event log and double click it. This will open a dialog box with the details of the event. To open the report, click the blue link.

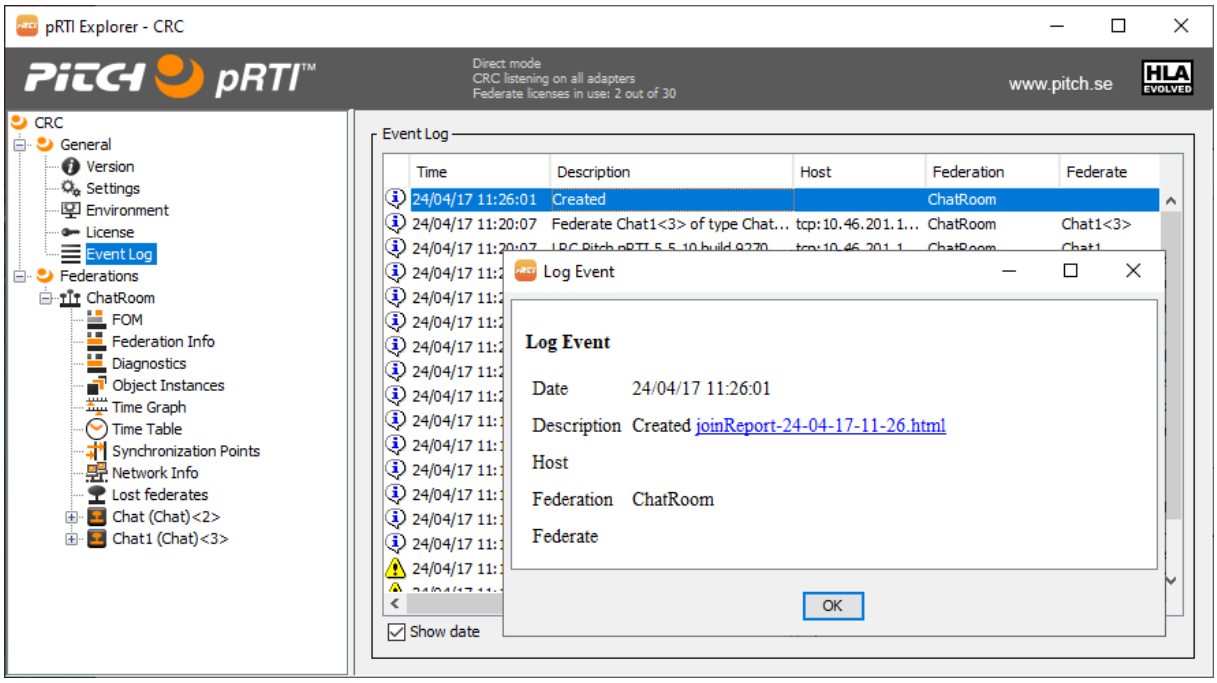

Figure 29. Report in Event log

5.2.10. Object Instances

The view shown in the figure below 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.

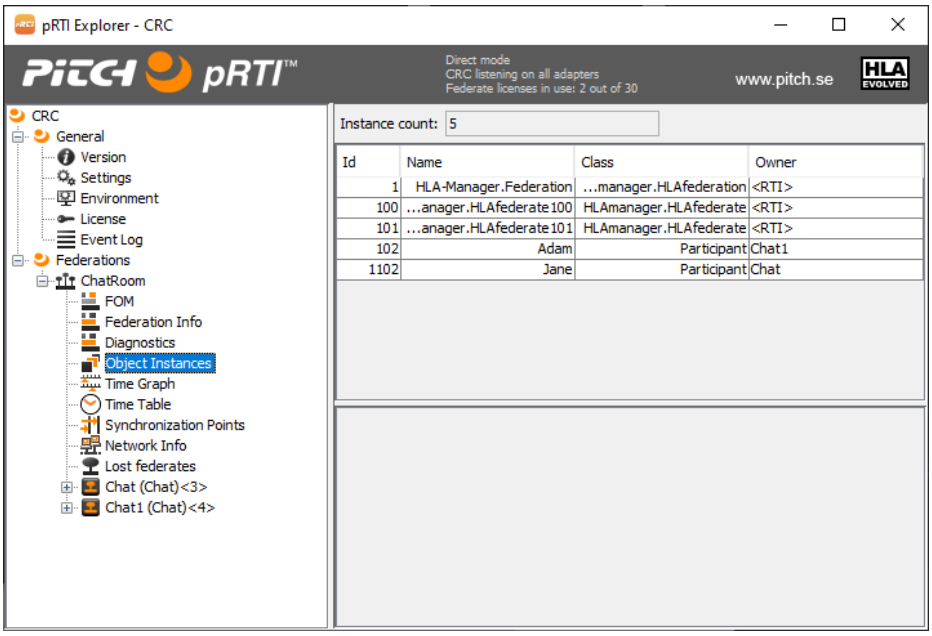

Figure 30. Inspection of object instances and attributes.

5.2.11. Time Graph

The figure below shows the time management state for the entire federation in the Time Graph view. 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.

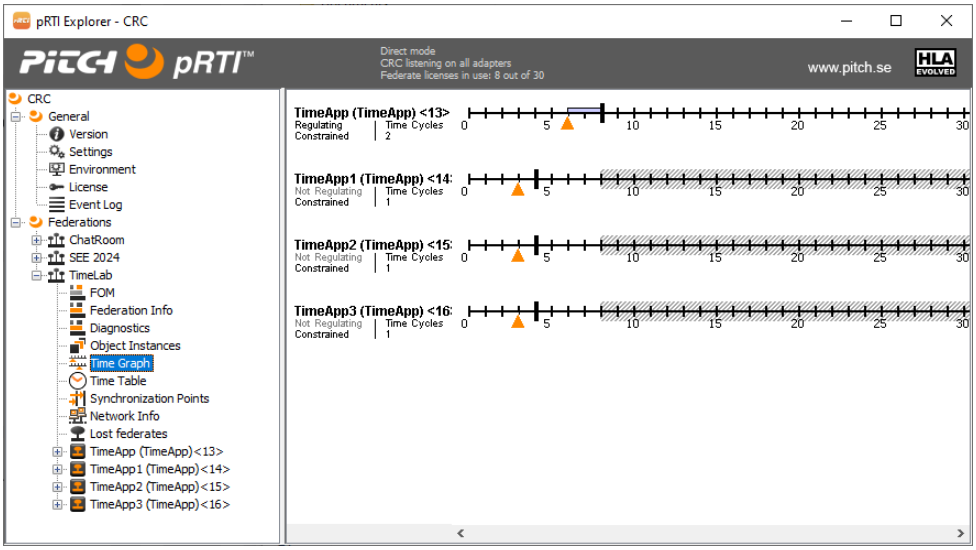

Figure 31. The time status for each federate in a graph view.

5.2.12. Time Table

The Time Table seen in the figure below shows an overview of logical time within the federation, including the current time of each federate, lookahead, whether the federate is

in the advancing state or the granted state.

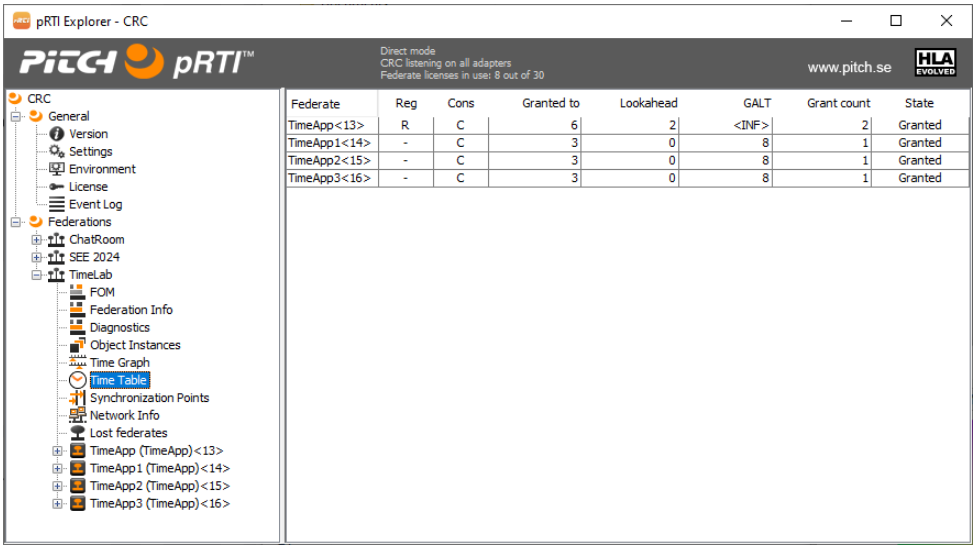

Figure 32. The time status for each federate in a table view.

5.2.13. Synchronization Points

The figure below shows the synchronization points that exist in the federation. This view will be empty if no synchronization points have been registered. The figure below shows three different synchronization points, and also the status for each federate in the federation. The synchronization point labeled *objects_discovered* has been achieved by the federate with federate id 4. The federates that are listed in the *Pending Federates Id* column have not achieved the synchronization point yet.

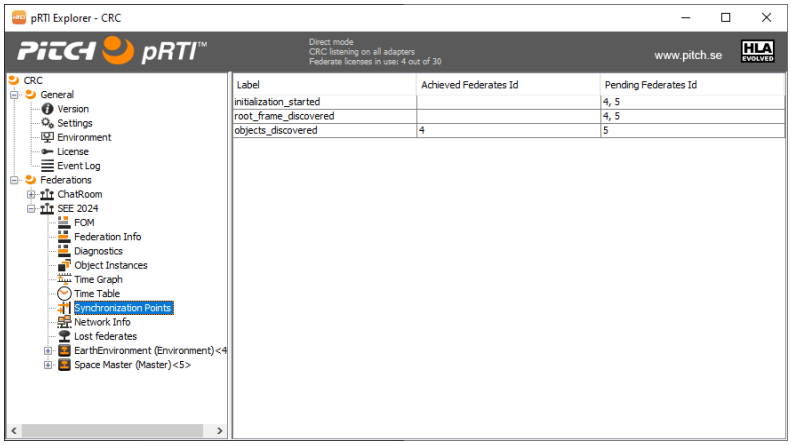

Figure 33. The synchronization points and federate synchronization status.

5.2.14. Network Info

The *Network Info* view lets you inspect the network links between the CRC and the LRCs in the federation. Clicking on a specific connection shows you statistics for that connections network traffic as shown in the figure below.

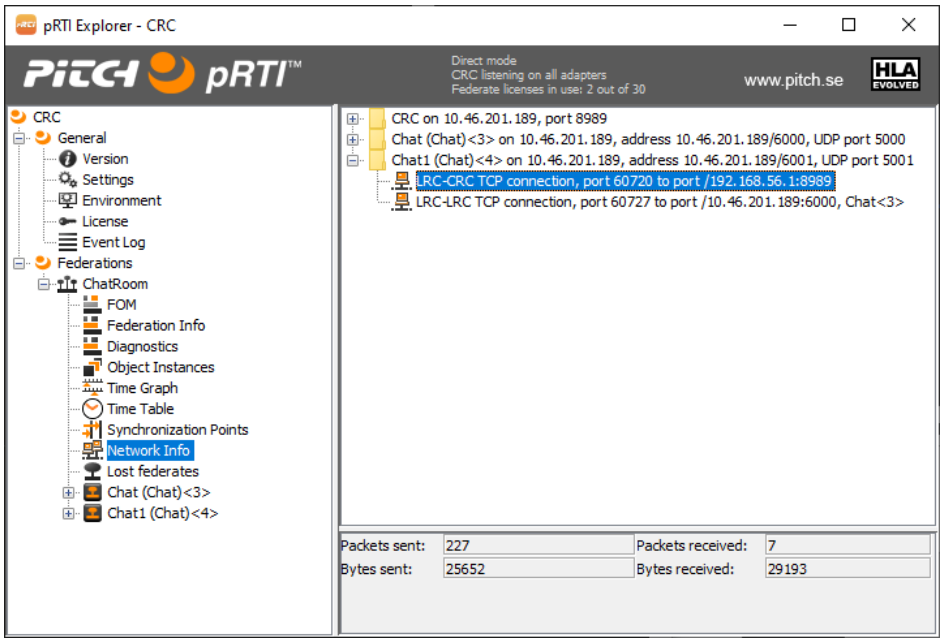

Figure 34. The network information for federation participants.

5.2.15. 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.

The figure below 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.

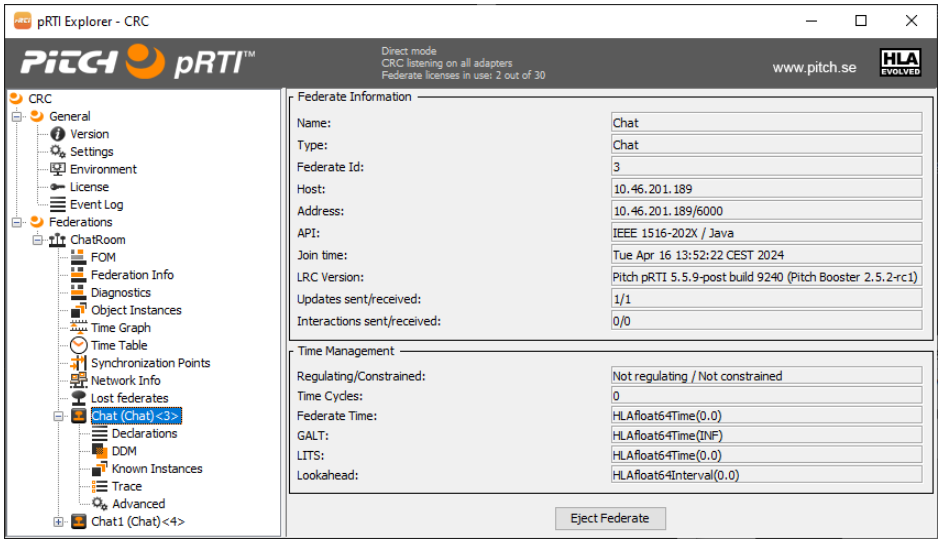

Figure 35. The federate information panel.

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 39 shows how to select specific services that should be traced.

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

The figure below 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.

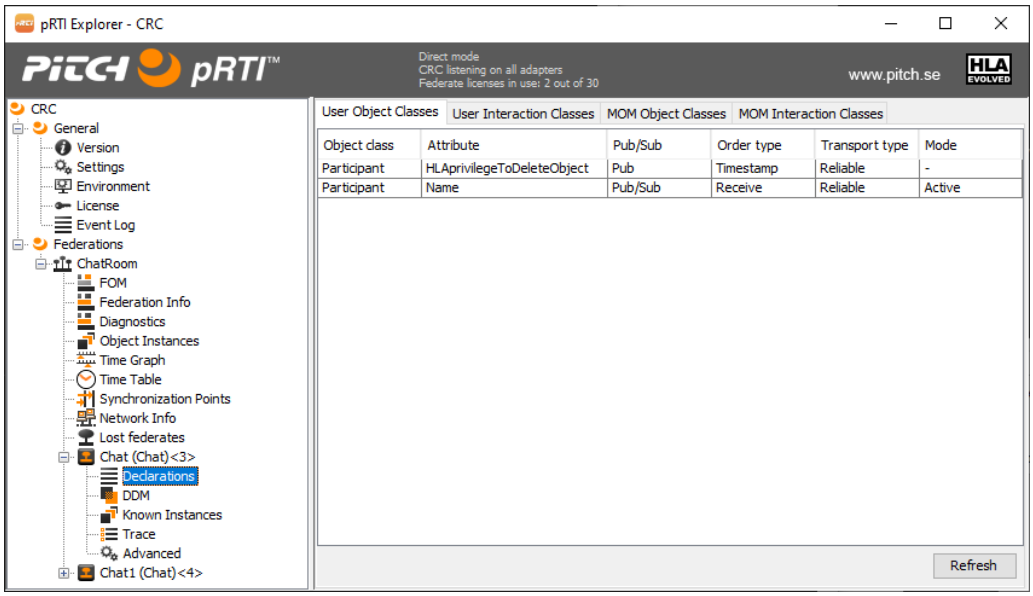

Figure 36. The object and interaction class declarations.

The figure below 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

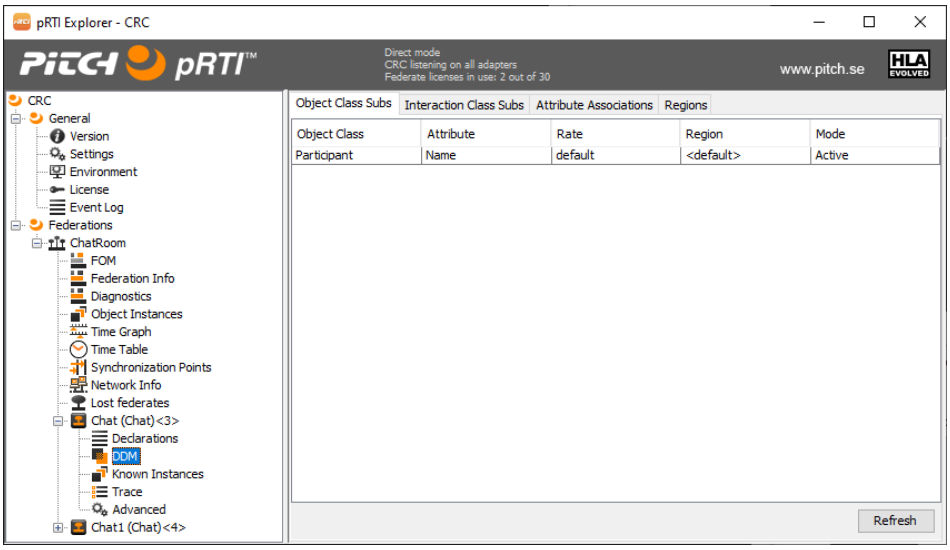

Figure 37. The DDM information.

The figure below 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.

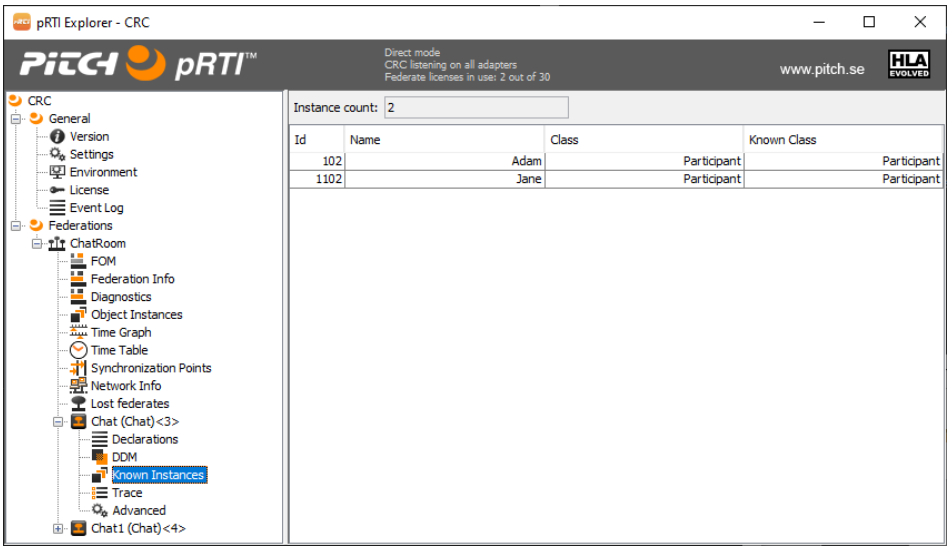

Figure 38. The discovered object instances.

The figure below 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.

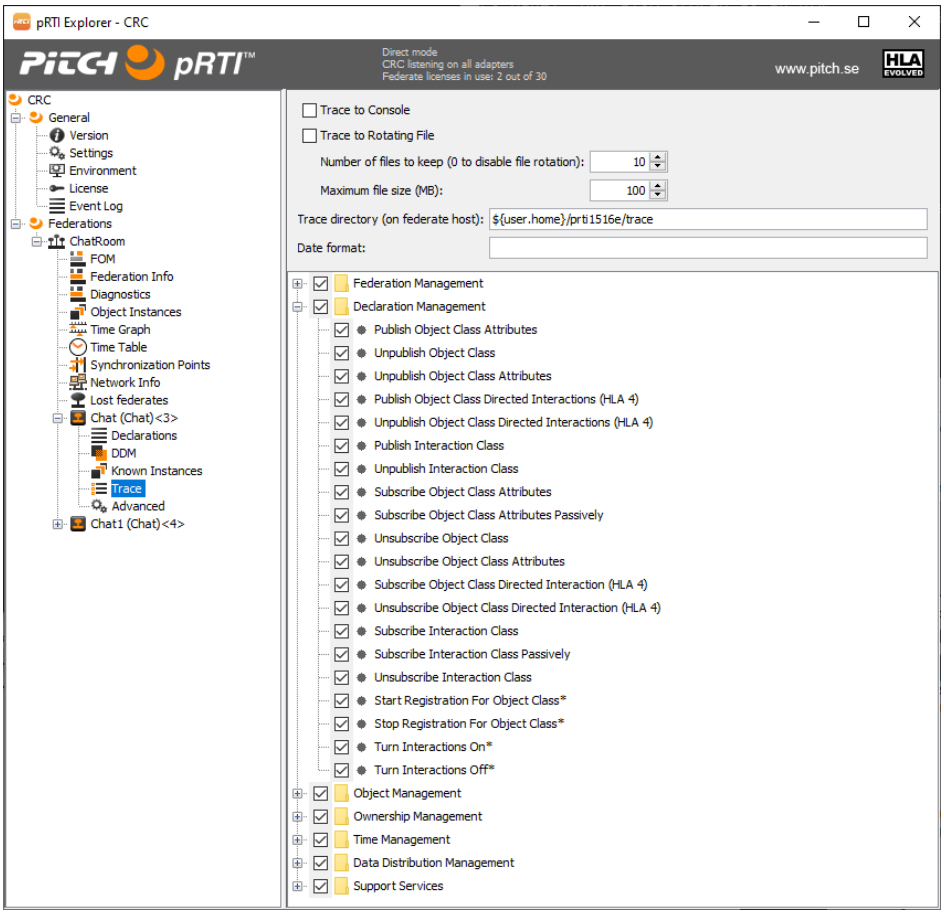

Figure 39. The tracing panel.

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.

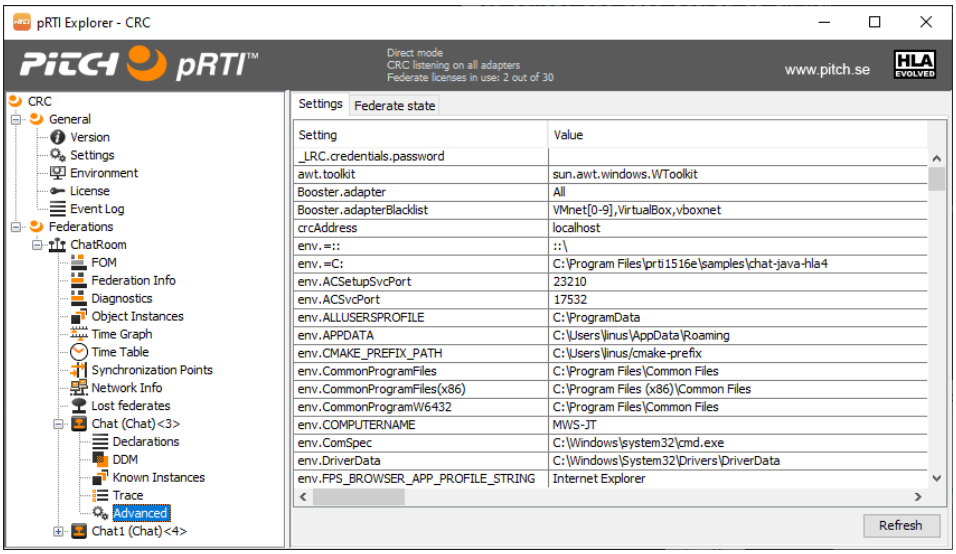

Figure 40. The advanced panel.

The figure above shows the advanced view where LRC diagnostics information as well as effective LRC settings can be inspected.

5.3. CRC startup options

The CRC is an application which may be started in the following modes:

- Interactive GUI application
- Interactive GUI application with command line prompt
- Interactive command line application
- Background service

There are different executable launchers for these startup modes. The CRC is a Java application, so some of the modes also have shell scripts invoking the java command as an optional way to start them and as a legacy from older versions.

Since the CRC is a Java application, it is possible to set parameters for the JVM running it. There are a number of generic Java parameters which can be set, and also a number of pRTI specific Java system properties which can be useful.

When launching the CRC using an **executable launcher** there are two ways to set JVM parameters:

Writing JVM parameters, each on a separate line, in the text file named `<executable launcher name>.vmoptions`

Adding each parameter as an argument to the launcher on the form `-J<parameter>`

When launching the CRC using a **shell script** which invokes the java command, the JVM parameters are added right after the java command in the script.

Some parameters are of the type which specifies Java system properties. Such parameters always start with `-D`. So, as an example when using the launcher argument to set a system property, the argument looks like
`-J-D<system property name>=<system property value>`

Any CRC setting value, which is available in the CRC settings file, can be set using system properties for the CRC launcher.

5.4. 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 [-v]`	Prints settings summary. Use `-v` argument to get all 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, 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 the figure below, is not displayed.

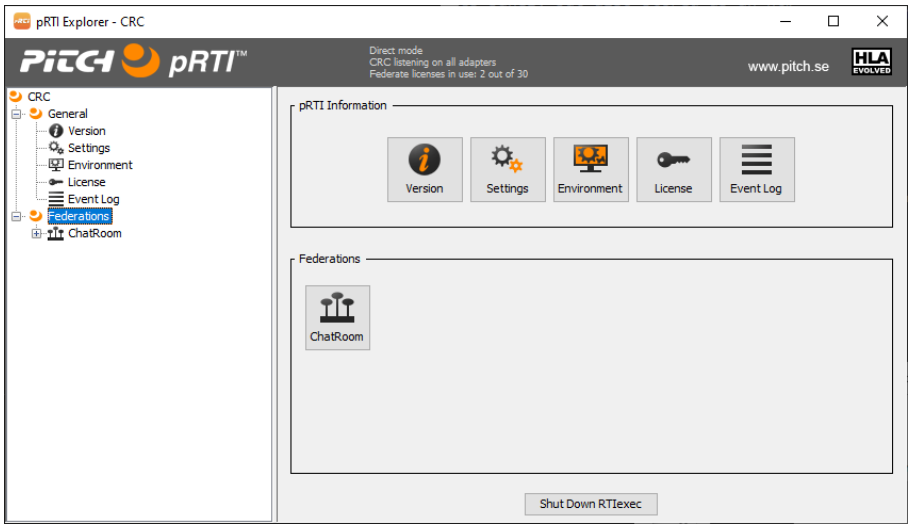

Figure 41. The pRTI™ Explorer.

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™ while 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_stderr.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.

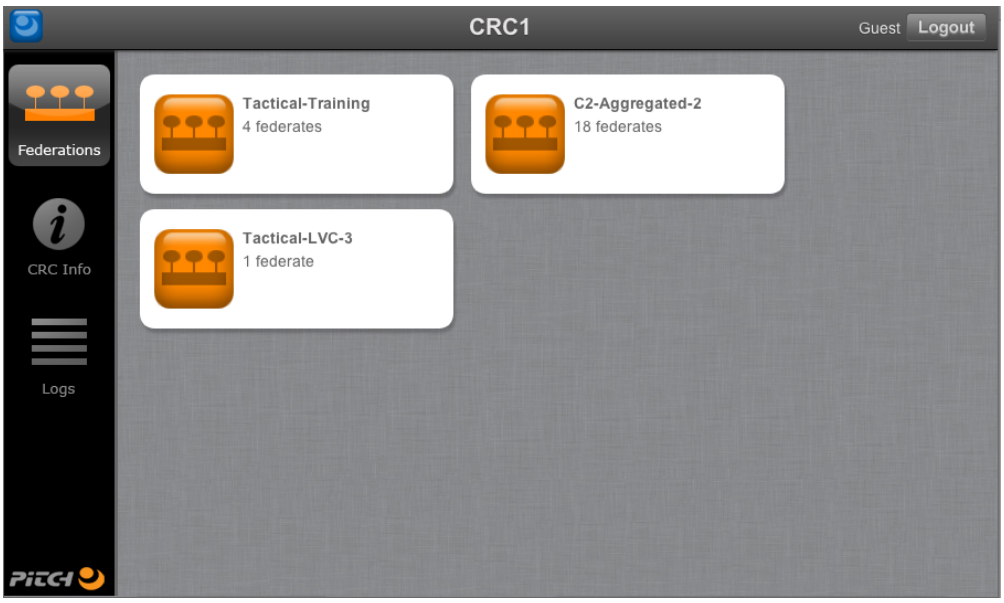

Figure 42. Pitch pRTI™ Web View.

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 described in Section 6, the Web View allows pRTI™ to be installed and deployed in a more robust manner, 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. Federate Protocol

Starting from version 5.5.8, pRTI™ introduces support for HLA 4's Federate Protocol, which is part of the IEEE 1516-202X standard. The Federate Protocol enhances flexibility and performance by enabling networked separation between federates and the RTI. It supports TCP and TLS transport protocols, offering options for both encrypted and unencrypted communication.

Some advantages of HLA's Federate Protocol are:

- Flexibility in cloud and container environments: Both federates and RTI can be vendor independent, allowing for greater adaptability in various infrastructures.
- Improved wireless connectivity: The Federate Protocol optimizes deployment by reducing federate handling of RTI traffic and enabling swift reconnection following brief disconnects between a federate and the server.
- Enhanced performance: Offloading federate RTI computations can alleviate CPU or memory bottlenecks on the machine running the federate, optimizing performance in specific scenarios.
- Security: With support for the TLS transport protocol, the Federate Protocol is suitable for security-sensitive environments.
- Simplified accreditation: The Federate Protocol streamlines the accreditation process for federates in secure environments and improves network communication inspection capabilities.
- Federates may be implemented in any programming language.

8.1. pRTI™ Federate Protocol architecture

pRTI™'s Federate Protocol architecture includes a server and optional client libraries.

The Federate Protocol Server bridges communication between federates and one RTI. Depending on your network design requirements, you can operate as many of these servers as needed.

Client libraries are supplied for simplified Federate Protocol federate development in C++ and Java. These libraries offer the standard HLA API to the federates and exchange Federate Protocol messages with the RTI.

We highly recomend using our client libraries for Federate Protocol federate development. If this isn't feasible, you'll need to create a custom Federate Protocol implementation to connect to Pitch pRTI™ Federate Protocol Server.

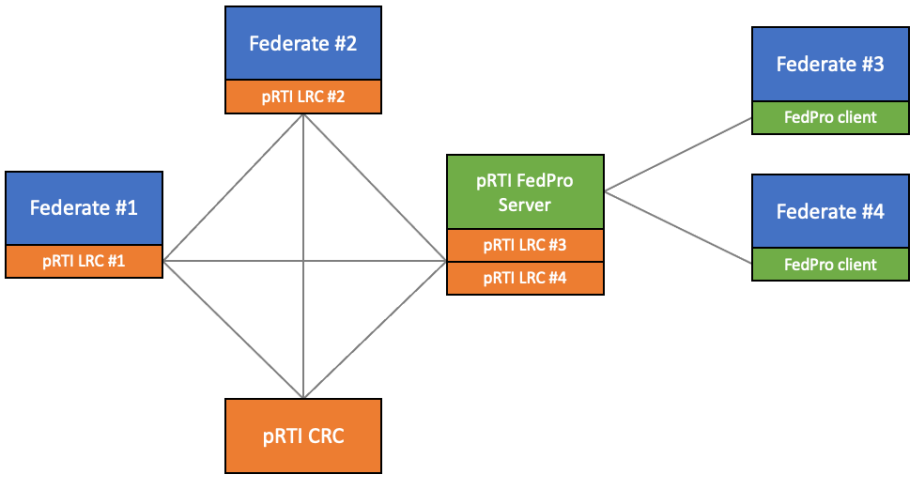

Figure 43. pRTI™'s topology including Federate Protocol federates.

8.2. Using Pitch pRTI™ Federate Protocol Server

The Federate Protocol Server is a standalone application that either can be run as a service or interactively through a command line user interface. For every client that connects to it, the Federate Protocol Server creates and maintains a separate connection to Pitch pRTI™.

For a client to be able to successfully connect to a federation execution through a Federate Protocol Server, a CRC also needs to be running just like for federation executions in which the Federate Protocol is not used.

8.2.1. Using the startup scripts

To start Federate Protocol Server from a command line, start the executable included in the pRTI™ directory.

For Mac and Linux machines, open a terminal and run the following command, followed by the applicable argument(s):

```
./<pRTI_directory>/bin/FederateProtocolServer [-a/--server-address <IP>] [-
f/--config-file <config_file>] [-h/--help] [-i/--interactive] [-p/--port
<port>] [-t/--timeout-heart <timeout seconds>] [-u/--timeout-purge <timeout
seconds>] [-w/--web]
```

For Windows machines, open a command prompt and run the following command, followed by the applicable argument(s):

```
<pRTI_directory>\bin\FederateProtocolServer.exe [-a/--server-address <IP>] [-
f/--config-file <config_file>] [-h/--help] [-i/--interactive] [-p/--port
<port>] [-t/--timeout-heart <timeout seconds>] [-u/--timeout-purge <timeout
seconds>] [-w/--web]
```

8.2.2. Command Line Arguments

The command line arguments that may be applied for the Federate Protocol Server are the following:

Option	Description
-h or -help	Displays the complete list of options.
-a <IP> or --server-address <IP>	Specify the host IP address on which the server should accept connections.
-f <file> or --config-file <file>	Read options from a specified configuration file.
-i or --interactive	Run the server through a command line user interface.
-p <port> or --port <port>	Specify which port the server will listen to. If specified, <port> should be replaced by an unsigned integer smaller or equal to 65 535. If not specified, the server will listen to the port 15164 if using communications protocol TCP, 80 if using communication protocol WebSocket.
-t <timeout> or --timeout -heart <timeout>	Specify <timeout> in whole seconds for when unresponsive client sessions shall be dropped. 0 indicates no timeout.
-u <timeout> or --timeout -purge <timeout>	Specify <timeout> in whole seconds for when dropped client sessions shall be purged and disconnected from pRTI™. -1 indicates no timeout, 0 indicates immediate purging.
-w or --web	Use WebSocket as communication protocol.

If multiple arguments are provided that would be conflicting, eg. when two different ports are specified on the command line, the argument specified last will be used.

8.2.3. Federate Protocol Server Configuration File

Federate Protocol Server supports using a configuration file to store common configurations in. The configuration file can be loaded by providing the command line argument `-f/--config-file` when starting the server.

The configuration file supports all arguments in their longer form that can be provided on the command line, the only exception being **config-file** itself. The settings in the configuration file follow the same defaults as the command line arguments. For example,

if a port is not specified in the configuration file, the Federate Protocol Server will default to using the standard port for the selected communication protocol. The following format should be used in the configuration file.

```
server-address=12.34.56.78
port=9876
```

8.2.4. Federate Protocol Server configuration

The Federate Protocol Server supports reading environment variables in addition to command line arguments and reading a configuration file. When using environment variables, they should be named on the format `FedPro.<option>`. All available command line arguments are possible to specify using environment variables.

In the case where conflicting options are provided in the different alternatives for specifying options, the Federate Protocol Server will choose the active option by the following priority list:

1. Command line arguments are of the highest priority
2. Environment variables are used when no corresponding command line argument is provided
3. Configuration file is used if there are neither a command line argument or an environment variable for the specified option

8.2.5. Federate Protocol Server Logging

The level of information to print to the console, to log and where to log, can be configured through a log configuration file. The default behavior is that the server will read the configuration from the file path `<pRTI_directory>/user.home/prti1516e/FedProServer.logging`. It is also possible to specify a custom log configuration file path in the file `<pRTI_directory>/bin/FederateProtocolServer.vmoptions`.

In the log configuration file, a file path which the server will log messages to can be specified. By default, this path is `<pRTI_directory>/logs/FedProServer`. To change the log level, specify the value of "se.pitch.fedpro.level" in the log configuration file. The available values are, from the least verbose to the most, `OFF`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST` and `ALL`. "INFO" is set to be the default log level.

8.3. Using Pitch pRTI™ Federate Protocol Client

Using a Federate Protocol Client requires providing the client with a slightly modified local settings designator, prefixed with `<Federate Protocol Server IP>[:<Federate Protocol`

`Server Port>];`. This prefix should then be followed by the usual local settings designator, but with addresses relative from the Federate Protocol Server. This is useful if, for example, the Federate Protocol Server is running on the same machine as the CRC. The local settings designator can then use 'localhost' instead of providing the actual address for the CRC. The IP for the Federate Protocol Server still needs to be provided. For example, if the Federate Protocol Server is located at `1.2.3.4:567`, and the server can access the CRC at `10.11.12.13:1415`, the local settings designator `argument=true` has to be set, the local settings designator that should be provided to the client is `1.2.3.4:567;10.11.12.13:1415;argument=true`.

8.3.1. Switching protocol used by Federate Protocol Client

There are multiple transport protocols that the Federate Protocol supports. The protocol that a federate will use can be provided in the local settings designator by using the setting `protocol=<Protocol Name>`. Alternatively, for Java clients, the following system property can be used as well `-DFedPro.default.protocol=<Protocol Name>`. TCP will be used as transport protocol if no option is provided.

Protocol	Protocol Name
TCP	tcp
WebSocket	websocket

8.3.2. The Federate Protocol Client Java Sample

The provided Federate Protocol Client Java sample is functionally identical to the provided HLA 4 Client Java sample. The provided bat/sh launchers start the sample with the -proto command line argument, this argument can also be used in the HLA 4 Java sample. The flag only changes which *RTIFactory* is loaded at runtime.

In the Federate Protocol Client sample, the loaded *RTIFactory* is named **Federate Protocol**. This factory is available when the client library is on the classpath. It is located in the lib folder in the installation root.

By running the client and not providing any local settings designator, the client will attempt to connect to a Federate Protocol Server and CRC running on localhost.

8.3.3. Identifying a client running though Federate Protocol

To identify that a client is communicating with pRTI™ through the Federate Protocol Server, enable 'Show Details' in pRTI™ Explorer and look below the Federate in question.

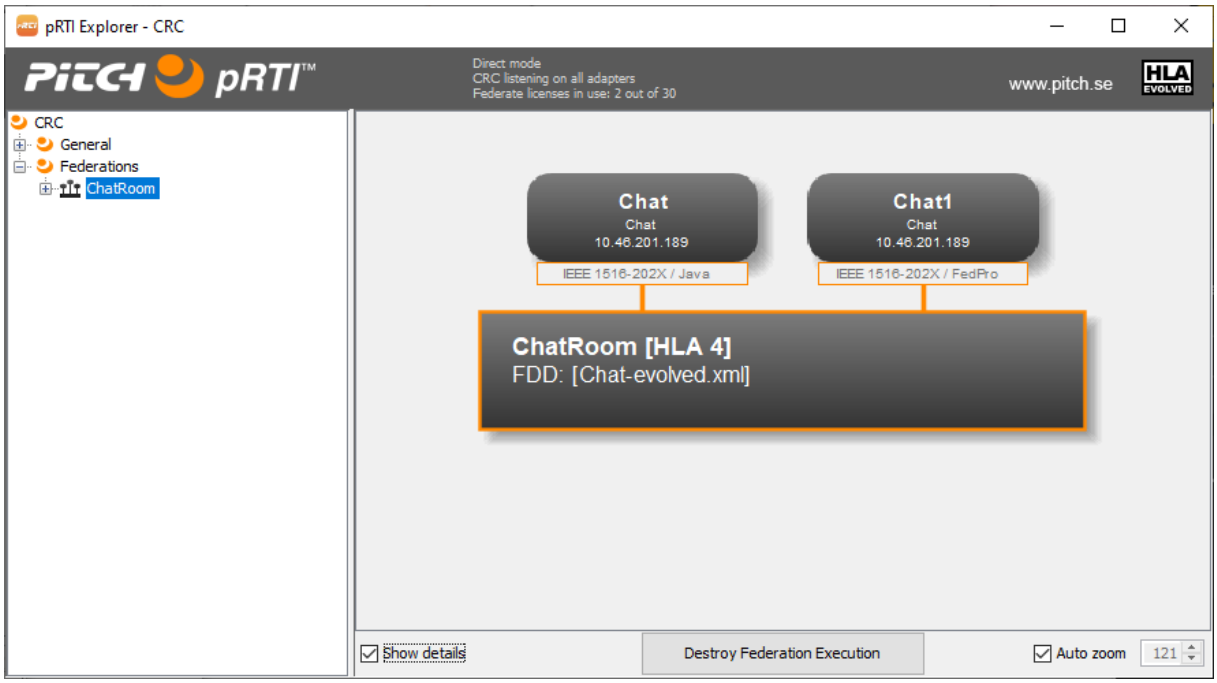

Figure 44. pRTI™ Explorer, with HLA 4 Sample and HLA 4 Federate Protocol Sample.

9. 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.

9.1. 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.

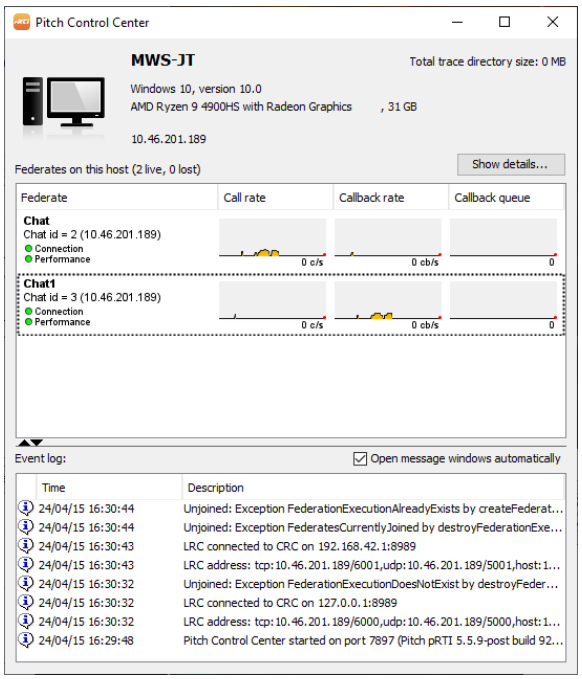

Figure 45. Pitch Control Center overview.

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.

9.2. 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.

9.3. 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 has 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 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.

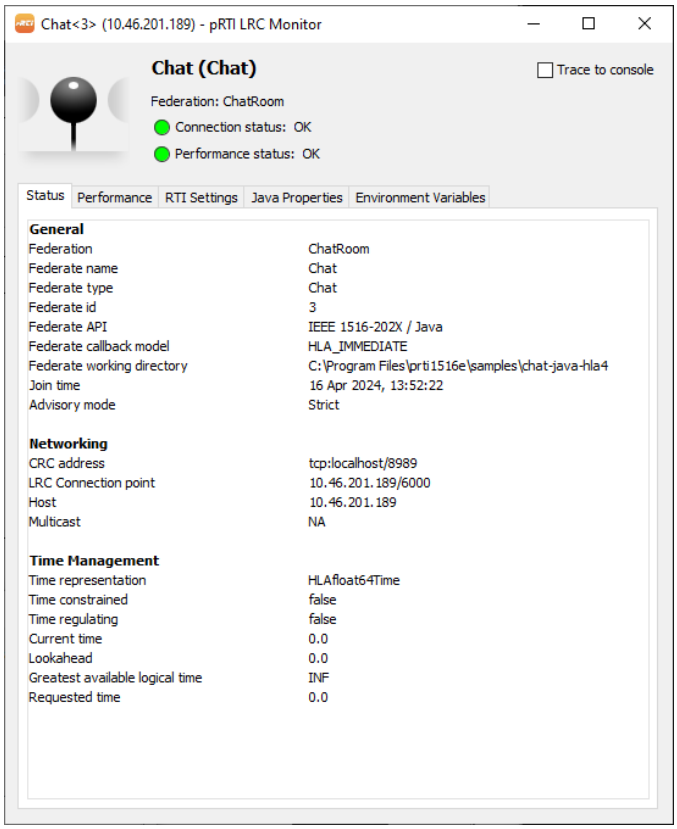

Figure 46. Pitch Control Center LRC monitor overview.

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.

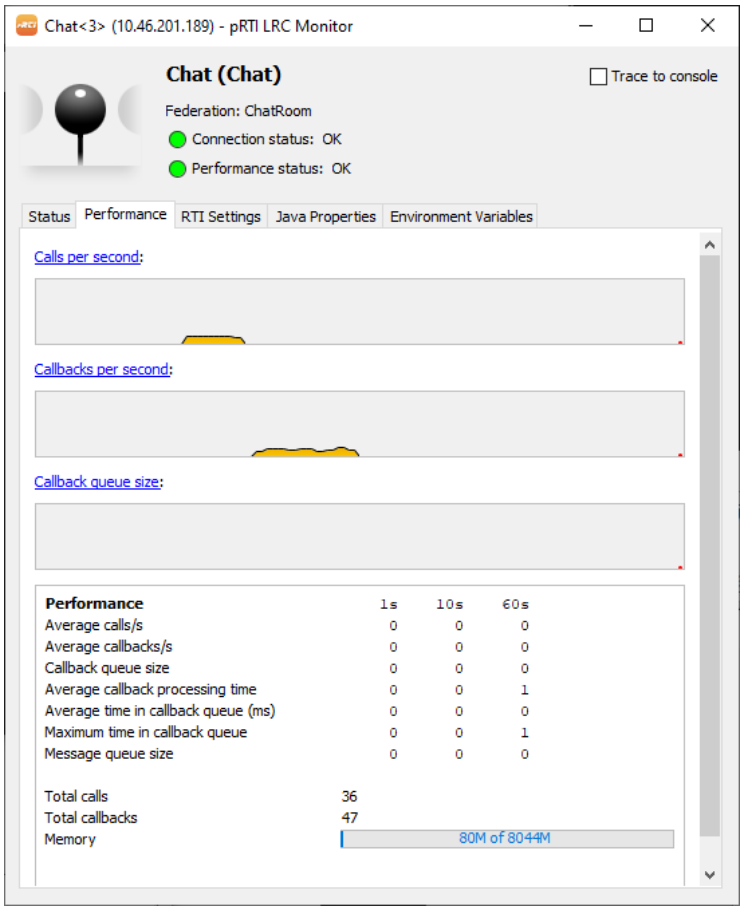

Figure 47. Pitch Control Center LRC performance monitor.

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.

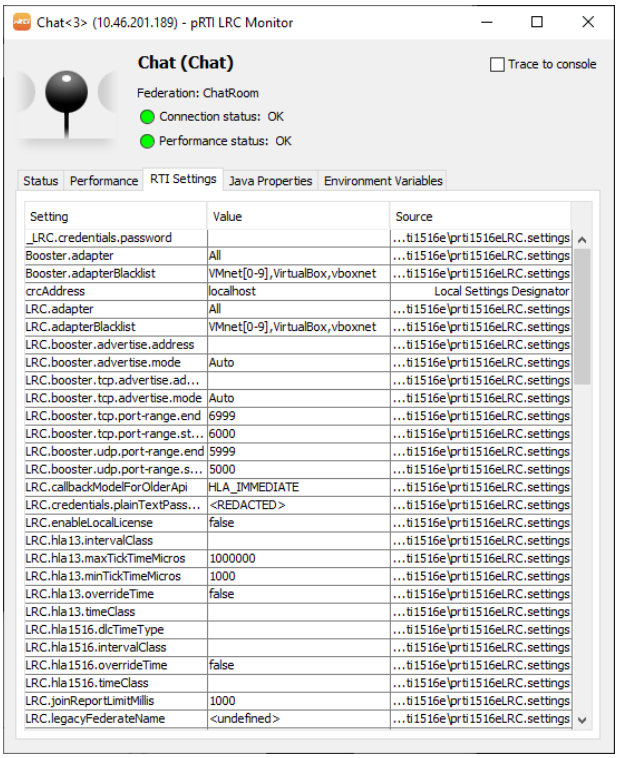

Figure 48. Pitch Control Center LRC monitor RTI settings.

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.

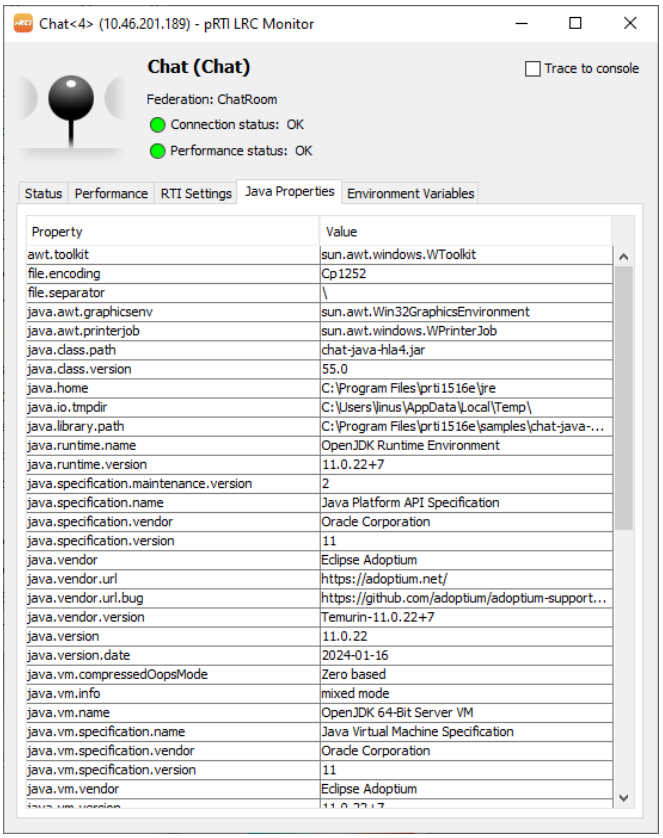

Figure 49. Pitch Control Center LRC monitor Java properties

The *Environment Variables* 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.

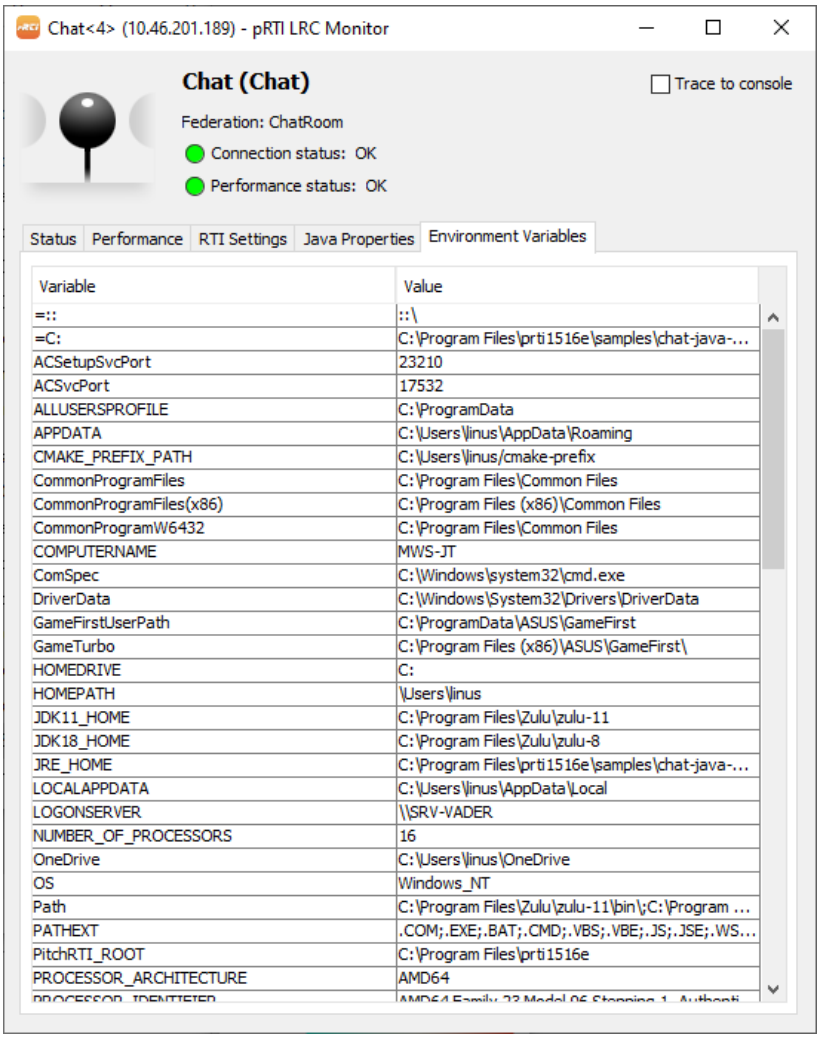

Figure 50. Pitch Control Center LRC monitor environment variables.

10. 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 DLLs 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™ website on www.pitchtechnologies.com 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 1.8.0 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.

10.1. Microsoft Visual Studio 2015 and above on Windows

10.1.1. Creating a New Project

Begin with creating a new project. Select the *Console App* template project_ for C++. Click **[Next]**.

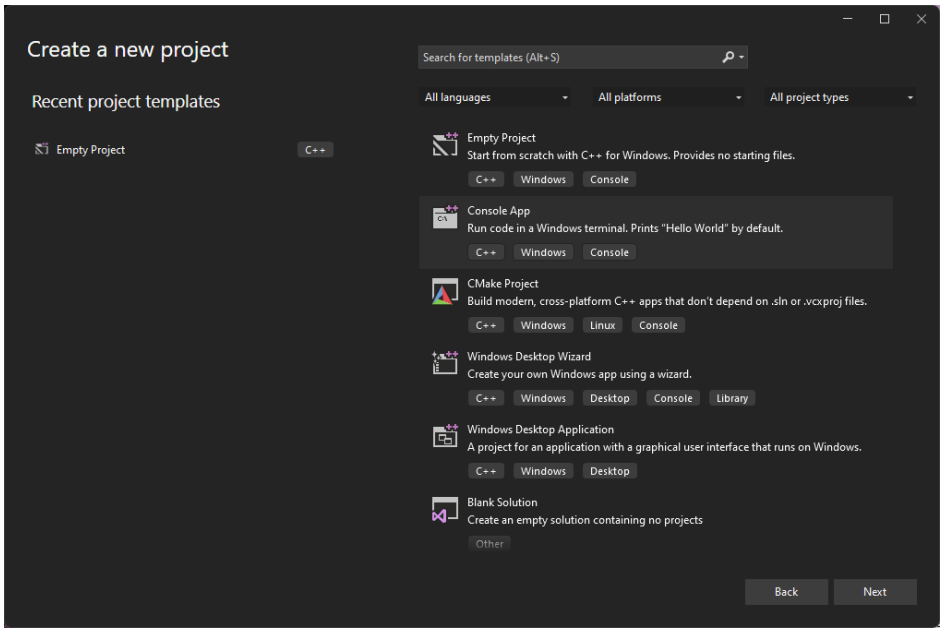

Figure 51. Creating a new project.

Next, name the project `chatcc`, place it in the `C:\Program Files\prti1516e\samples` directory, and check *Place solution and project in the same directory*.

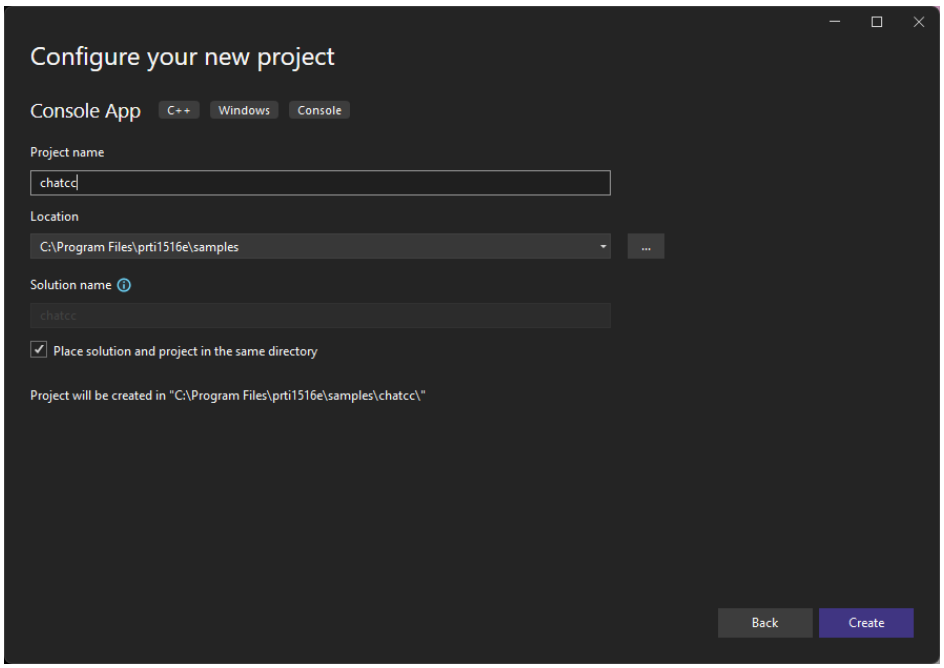

Figure 52. Application settings for the chatcc project.

Copy the source directory `C:\Program Files\prti1516e\samples\chat-cpp-1516e\src` to your new project directory `C:\Program Files\prti1516e\samples\chatcc`.

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

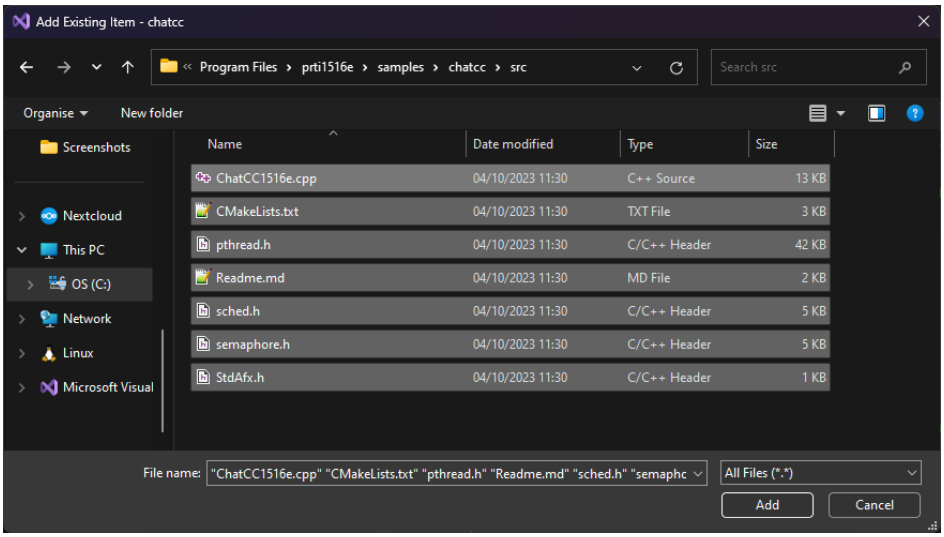

Figure 53. Add source files to the project.

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.

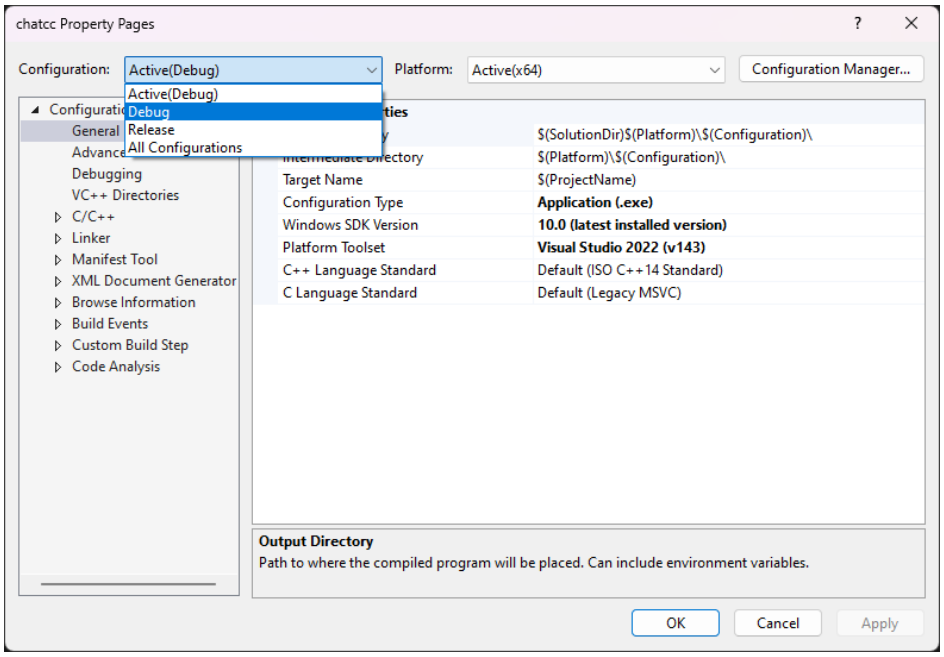

Figure 54. Settings for the Debug configuration.

10.1.2. 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\chatcc\src`.

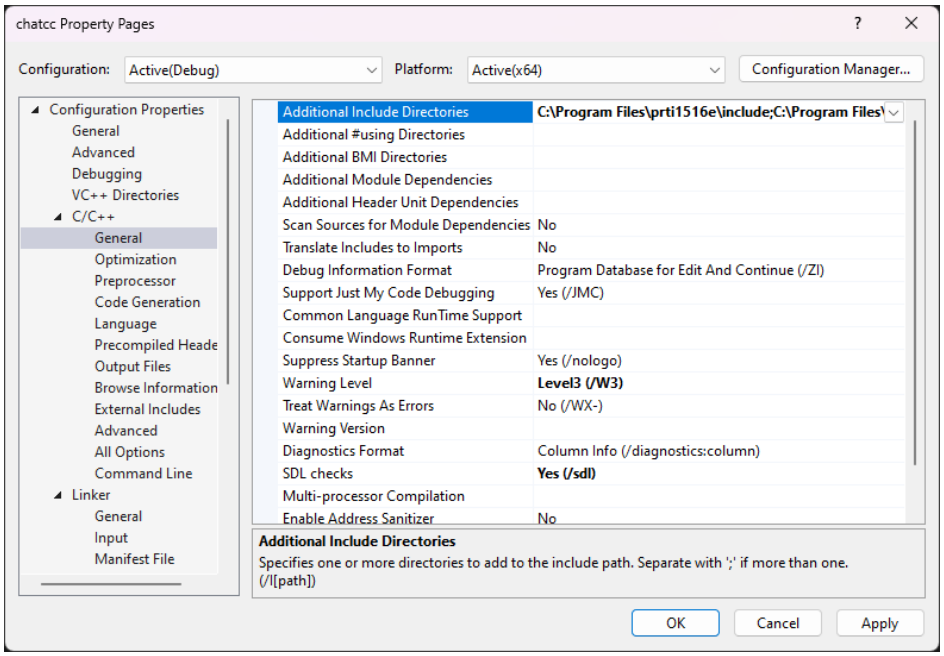

Figure 55. Specify additional include directories.

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.

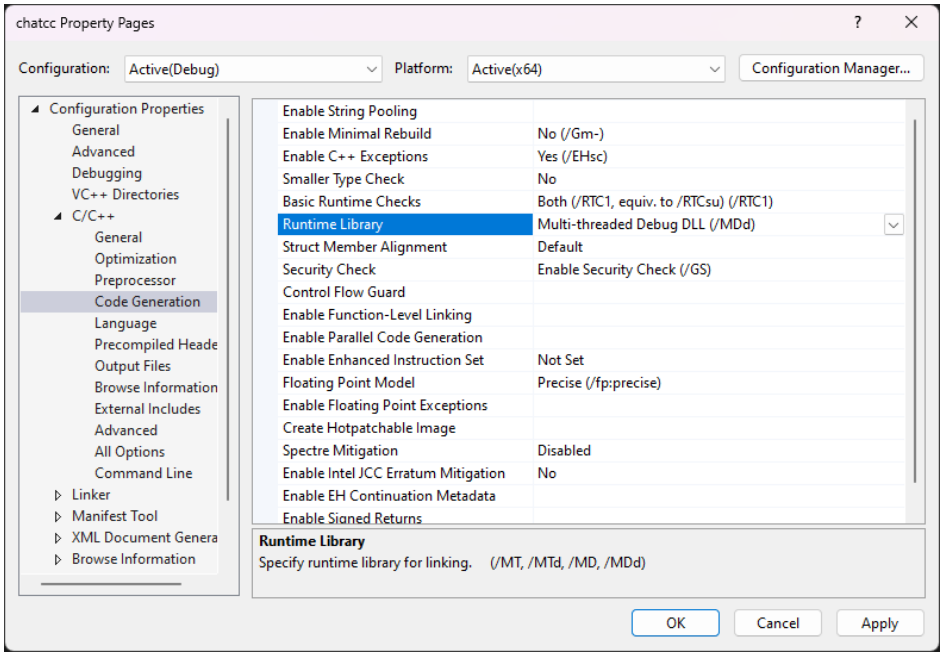

Figure 56. Select run-time library to use.

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

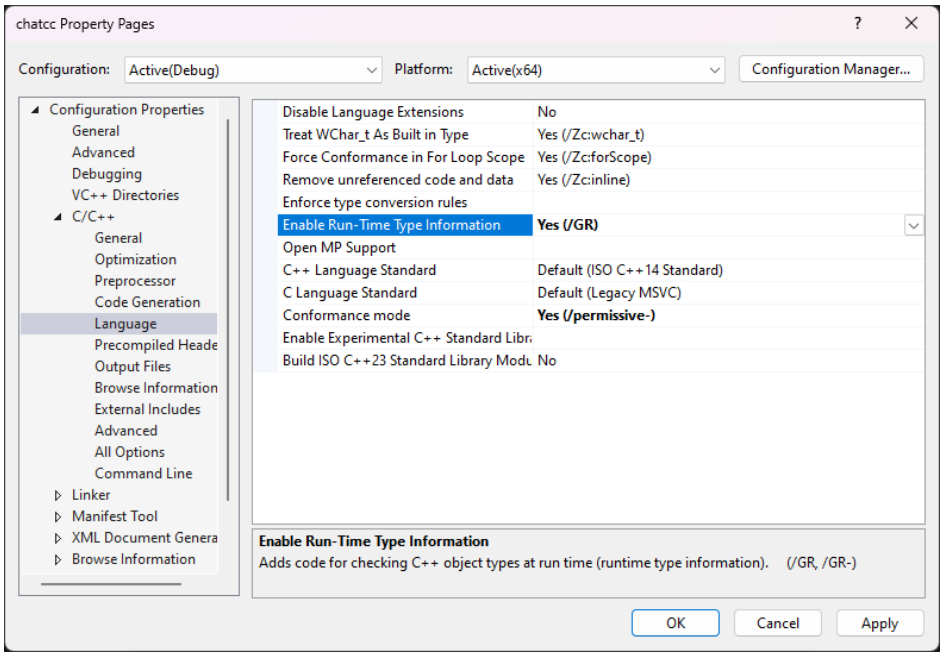

Figure 57. Enable Run-Time Type Information (RTTI).

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.

10.1.3. 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™. For 64-bit programs built using Visual Studio 2015 and above, this is `C:\Program Files\prti1516e\lib\vc140_64`. You will also need to add the directory containing the *pthread* library: `C:\Program Files\prti1516e\samples\chat-cpp-1516e`.

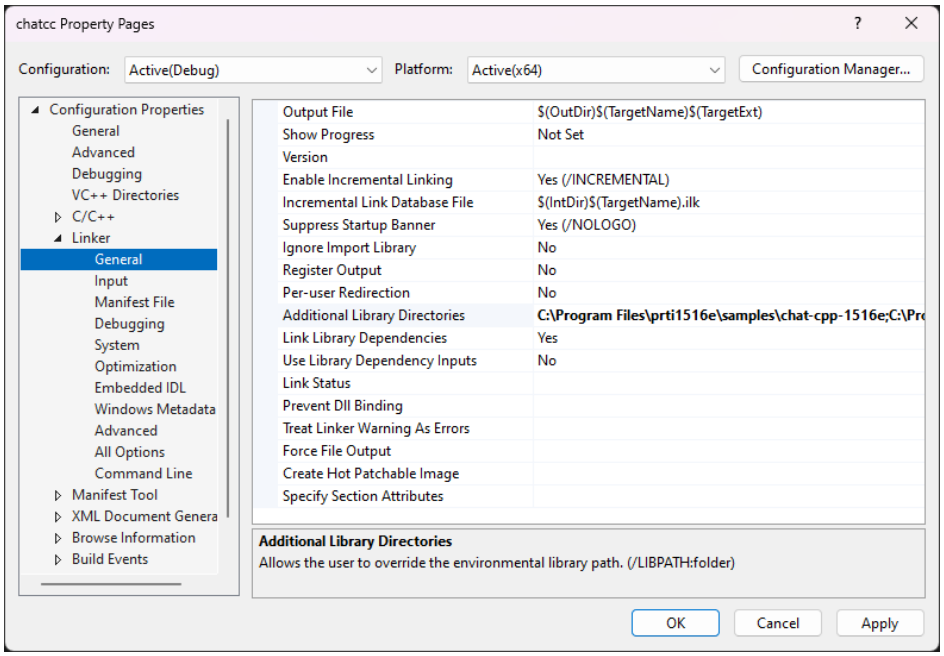

Figure 58. Specify additional library directories.

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.

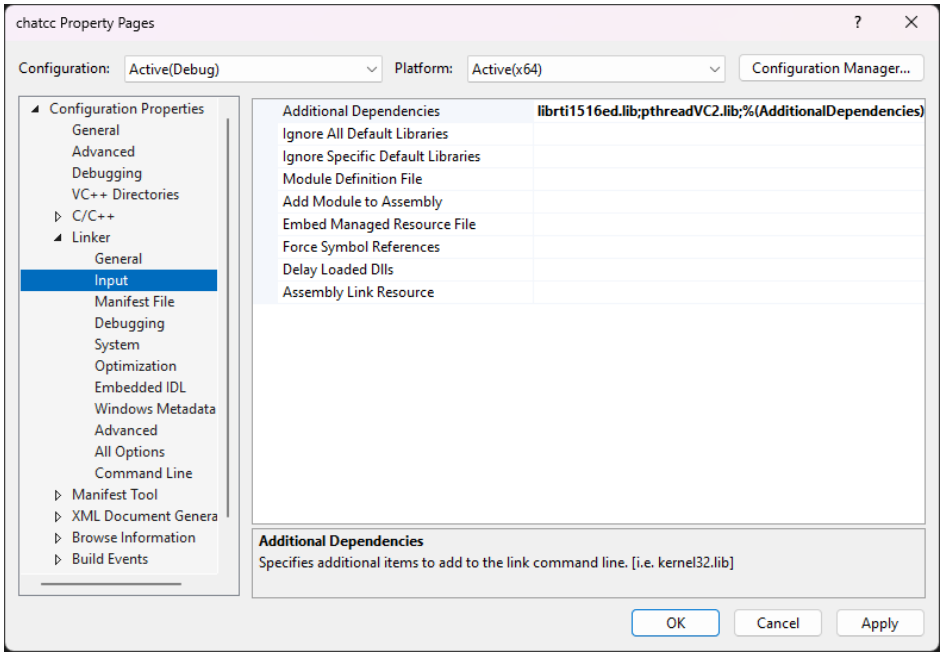

Figure 59. Specify additional dependencies.

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.

10.2. 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`.

10.3. 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, 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.

10.4. Other Microsoft Visual Studio versions on Windows

The project configuration steps described for Microsoft Visual Studio 2010 are similar to the configuration of the other supported Visual Studio versions - from 6.0 and .NET 2003, to 2019 and 2022.

10.5. Using HLA 4 with Visual Studio

To use the new HLA 4 Preview in C++ with Visual Studio, use the libraries:

- librti1516_202X<platform>.lib - `$PRTI_HOME/lib`
- libfedtime1516_202X<platform>.lib - `$PRTI_HOME/lib`

Where `<platform>` refers to the compiler being used, e.g. `vc140` for Visual Studio 2015.

Include files for HLA 4 are found in the `HLA4` subdirectory of the pRTI include files directory. In the default installation, this is `C:\Program Files\prti1516e\include\HLA4`.

10.6. 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.

10.6.1. Modifying the Makefile

In the `src` directory found in the `samples/chat-cpp-1516e` directory contains a *Makefile* which can be used to build the Evolved chat example federate. Run *make* to build the federate.

10.6.2. Adding prti1516e.jar to the CLASSPATH

To be able to use pRTI™, you need to ensure that the `prti1516e.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 prti1516e.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 prti1516e.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.

To use the new HLA 4 Preview, the file you need on the CLASSPATH is `prti1516_202X.jar`.

10.6.3. 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`

And the similar libraries for 64-bit systems are:

- librti1516e.so - `$PRTI_HOME/lib/gcc41_64`
- libfedtime1516e.so - `$PRTI_HOME/lib/gcc41_64`

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.

To use the new HLA 4 Preview in C++, use the libraries:

- librti1516_202X<platform>.so - `$PRTI_HOME/lib`
- libfedtime1516_202X<platform>.so - `$PRTI_HOME/lib`

Where `<platform>` refers to the compiler being used, e.g. `gcc7` for gcc 7.

10.7. 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.

10.8. 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.

To use the HLA 4 Preview with Java, you need the file `prti1516_202X.jar` on your CLASSPATH.

11. 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.pitchtechnologies.com. It is assumed that you are familiar with the contents of chapter 6 from the tutorial. 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 the figure below.

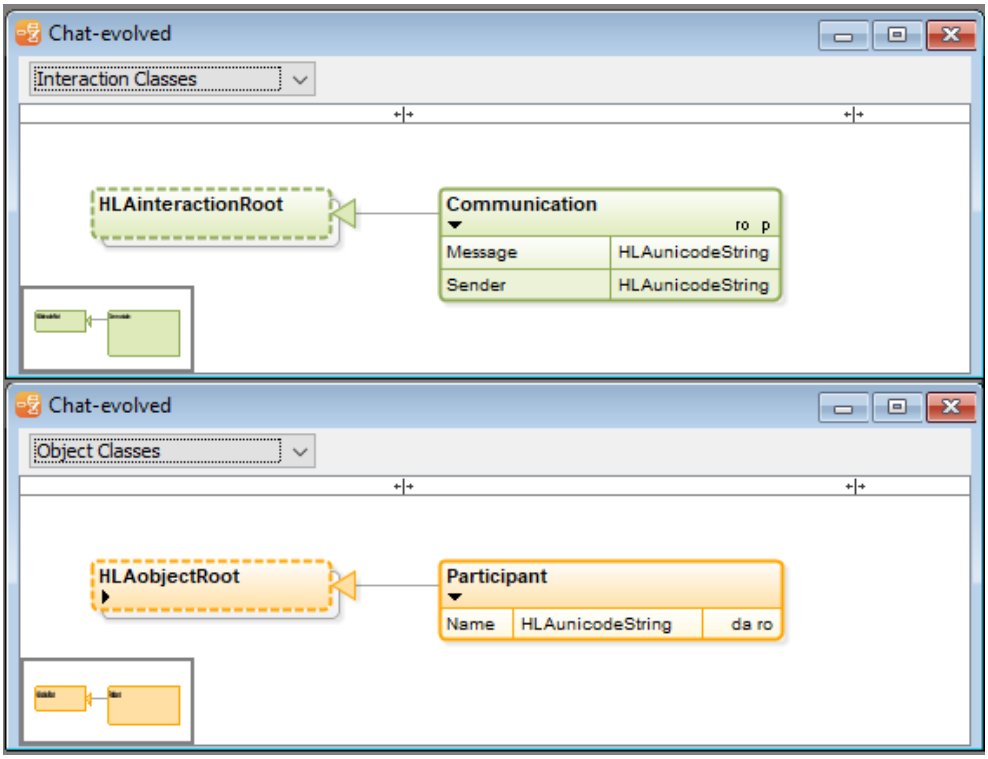

Figure 60. The FOM used in the ChatRoom federation.

11.1. 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.

11.2. 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.

11.3. 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 e) {
}
```

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 to 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.

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. 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).

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;
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.

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. 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*.

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 www.pitchtechnologies.com.

12. 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.pitchtechnologies.com. It is assumed that you are familiar with the contents of chapter 6 from the tutorial. 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 the figure below.

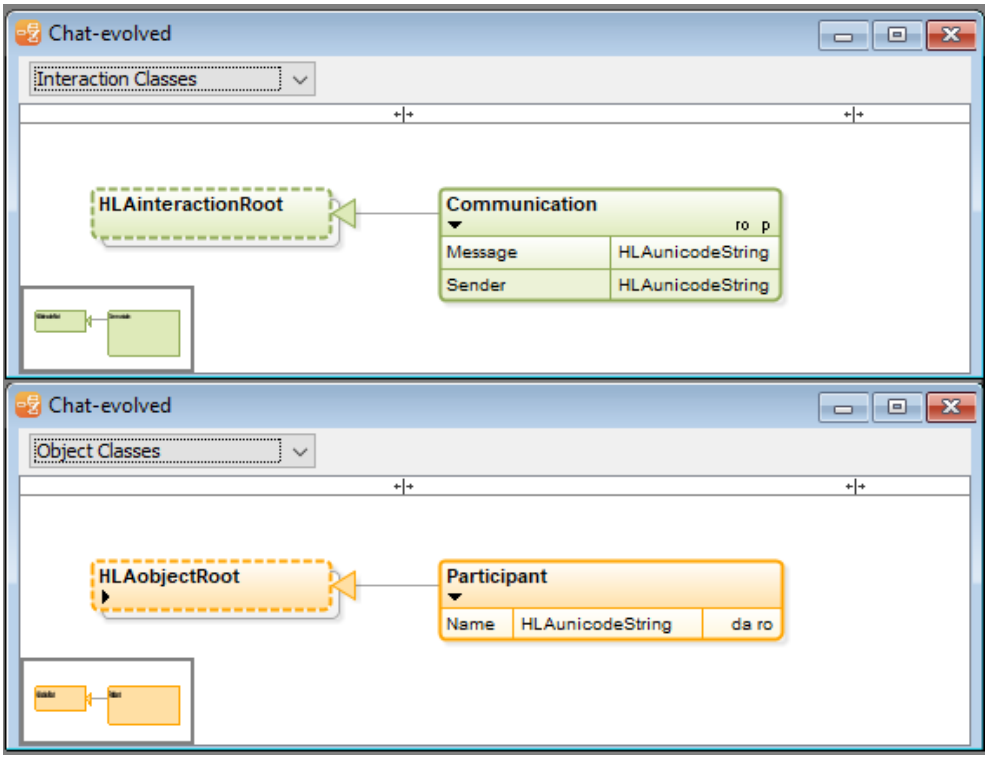

Figure 61. The FOM used in the ChatRoom federation.

12.1. 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 RTIs that are 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.

12.2. 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.

12.3. 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.

12.4. 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).

12.5. 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.

12.6. 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*.

12.7. 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 www.pitchtechnologies.com

13. 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.

13.1. 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 multithreaded 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()`.

Regardless of process model chosen, pRTI processes concurrent service invocations to the same RTIambassador instance sequentially.

13.2. Practical Guidelines

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

- **For the fastest porting:** Disable multithreading 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.

13.3. 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 www.sisostds.org.

14. Debugging and Tracing

14.1. 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:

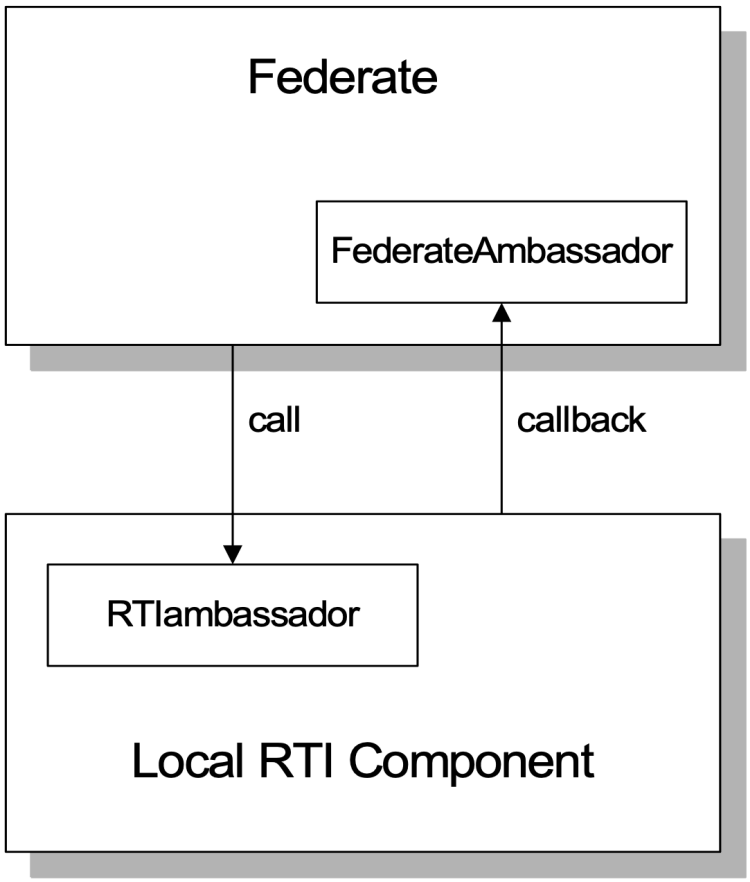

Figure 62. Calls and callbacks.

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, for example Pitch Recorder(tm).

14.2. 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.

14.3. 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.

14.4. 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 3.4 in this manual.

Notice the `receiveInteraction` in the final line with the hex code 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))
```

15. 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 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 RTIs 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 multithreaded federates instead of ticked federates. Note that a federation may mix the two types of federates. Read more in Section 13.

15.1. 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.

15.2. Overview of Pitch pRTI™ Communication

When the pRTI™ components (CRC and LRCs) 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 LRCs 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 the figure below shows.

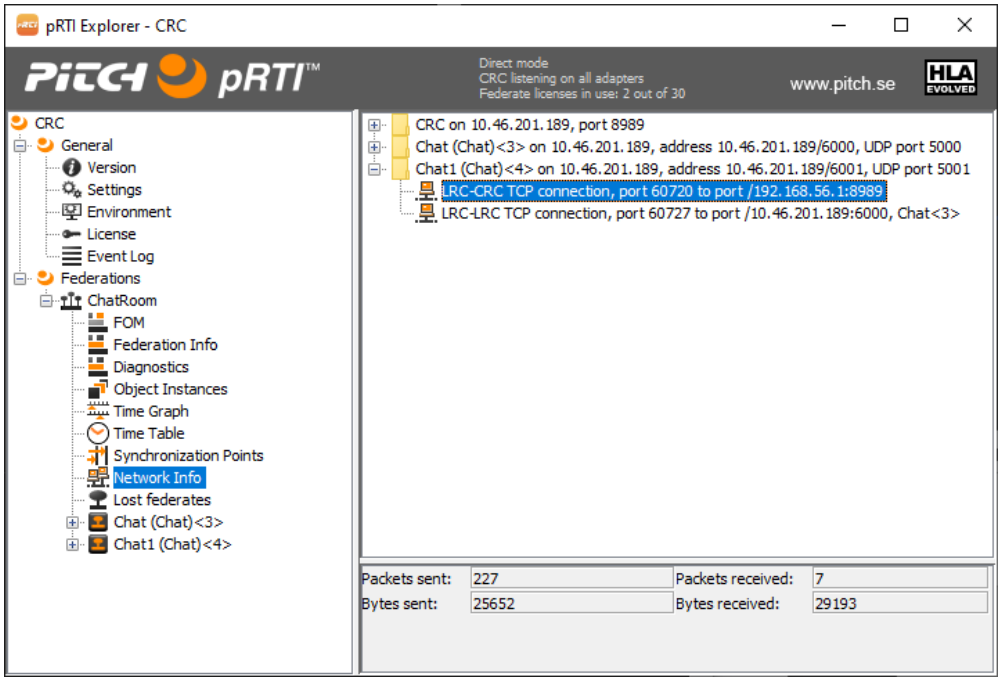

Figure 63. The network connections between the LRCs and the CRC.

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 LRCs.

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 LRCs 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 17.1 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 17.2 for more information.

15.3. 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.

15.4. 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.

By default, the CRC opens port 1099 plus an anonymous port for Web View. Since an anonymous port cannot be opened in a firewall, there is a setting to set a specific port.

Let's say that we want pRTI to use port 55555 for Web View. To accomplish this, add the following setting to `prti1516eCRC.settings`:

```
prti1516e.remoteGui.port=55555
```

15.5. Network Settings

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

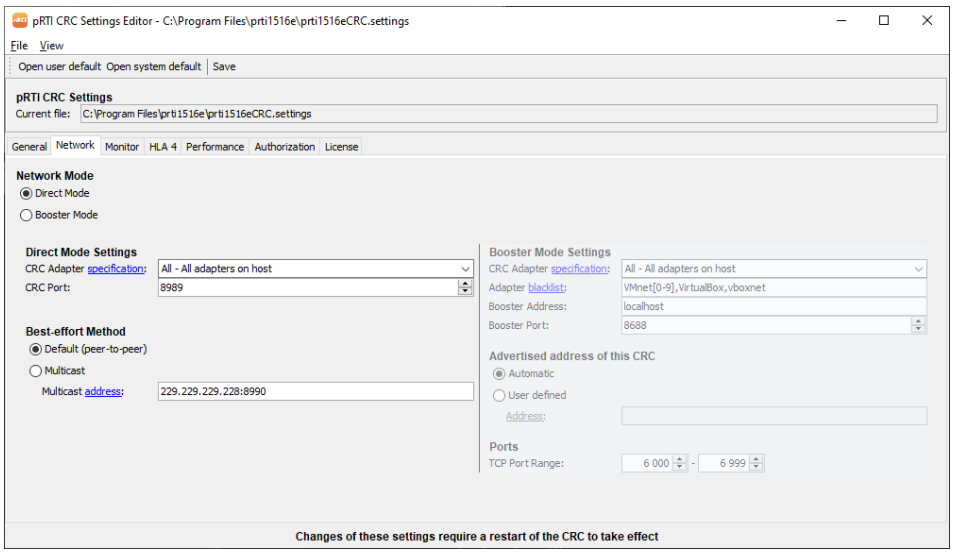

Figure 64. Changing the settings for the CRC.

See Section 17.1 for more information on changing the CRC settings.

For the LRC the network settings are available in both the LRC GUI shown in the figure below and in a settings file.

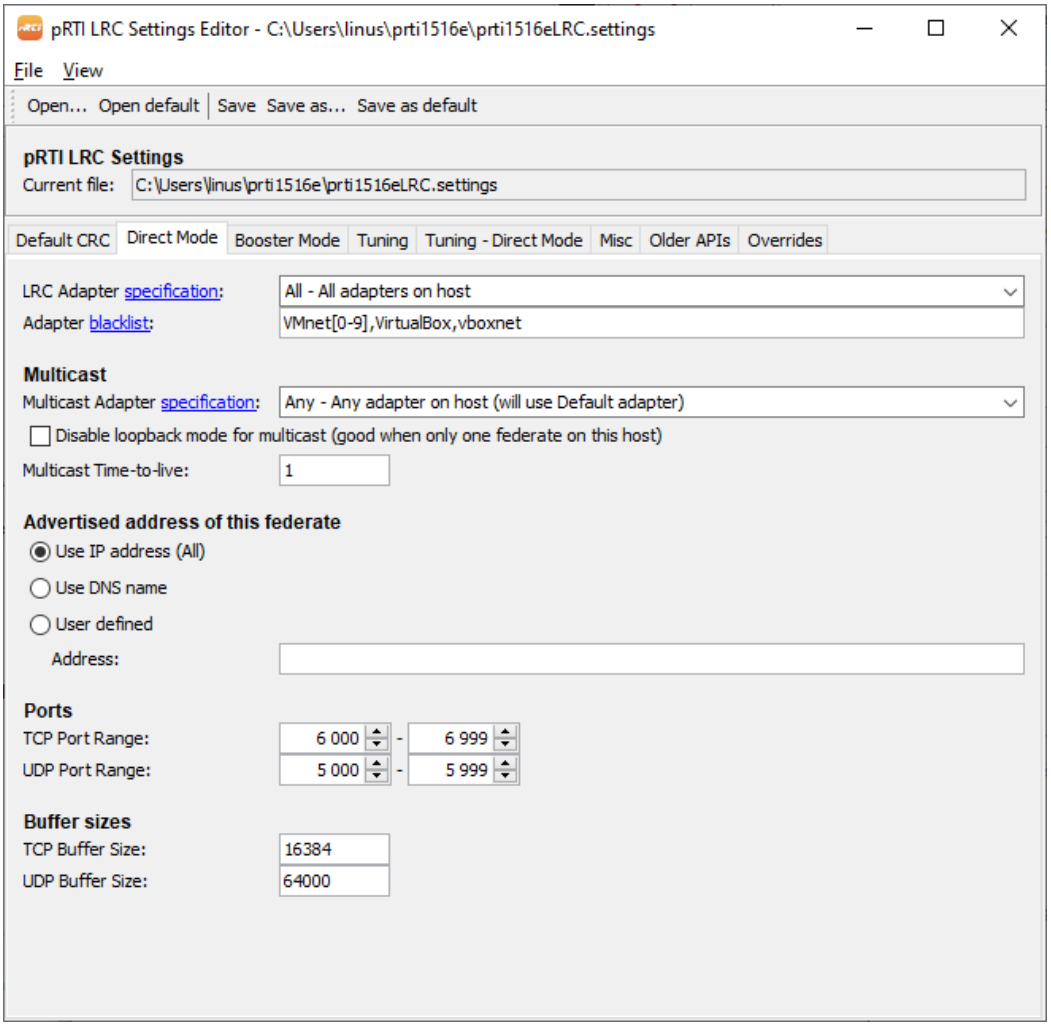

Figure 65. Changing the settings for the LRC.

See Section 17.2 for more information on changing the LRC settings.

15.6. 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.

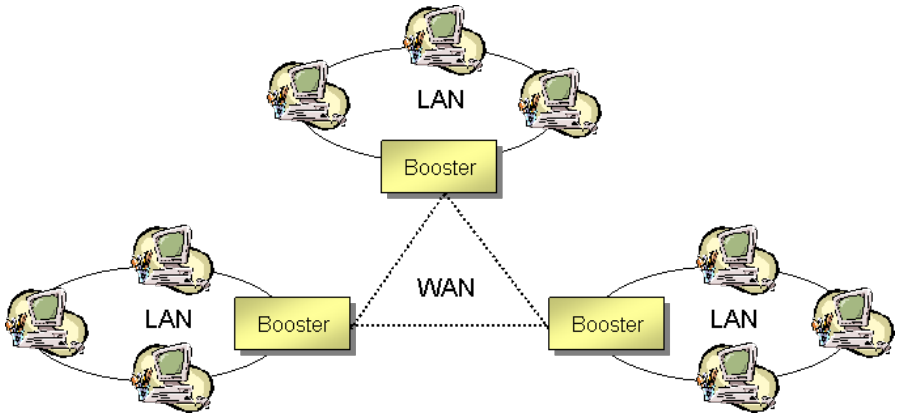

Figure 66. The Pitch Booster™ topology.

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 send 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.

15.6.1. Configuring the LRC and CRC for Pitch Booster™

To enable the use of the Pitch Booster™, open the CRC Settings tool, go to the *Network* 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
```

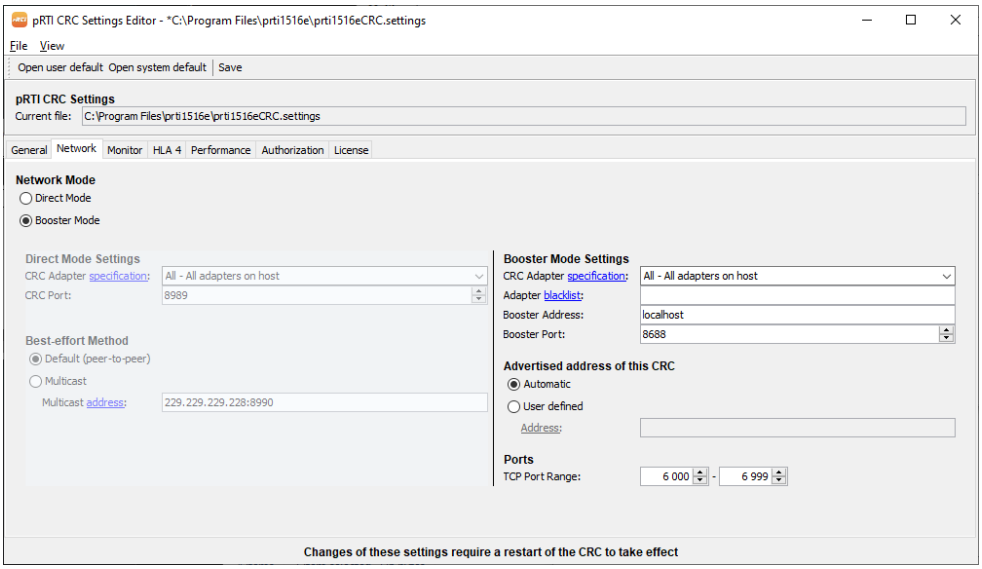

Figure 67. Enabling Pitch Booster™ through the CRC Settings.

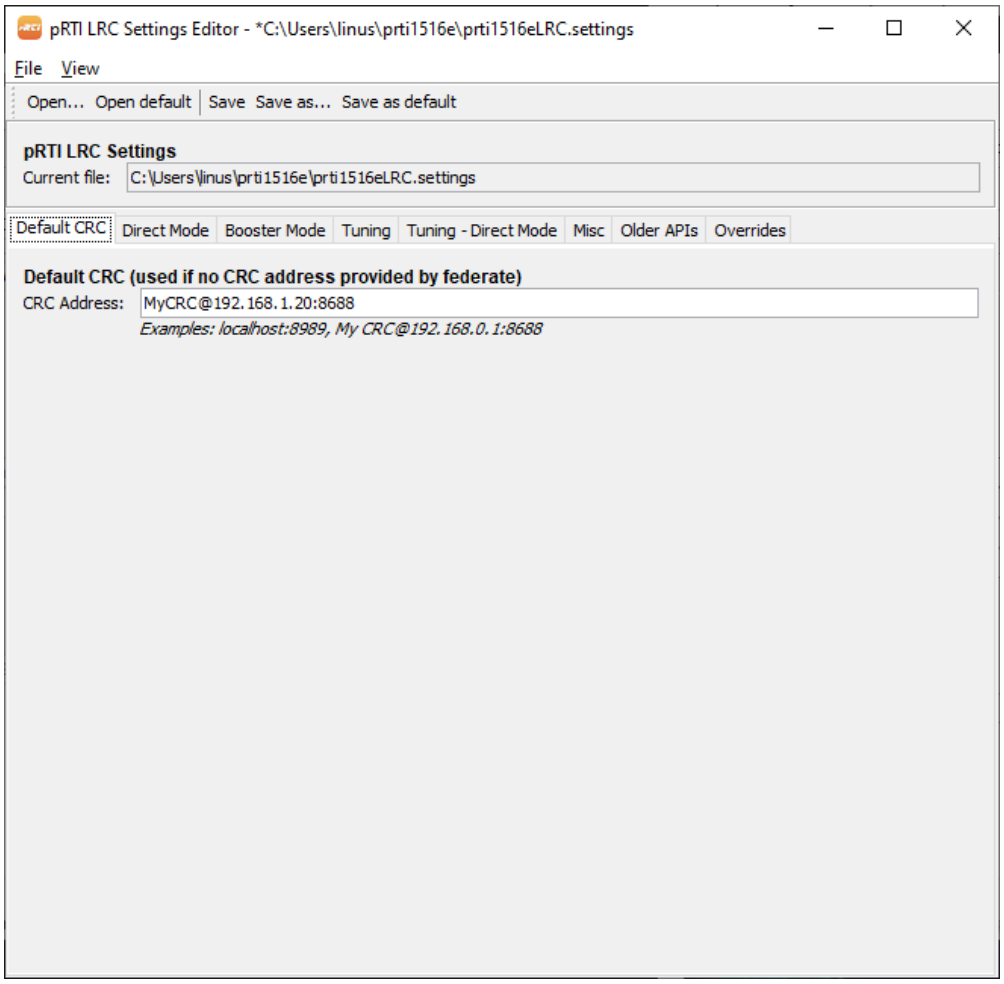

Figure 68. Enabling Pitch Booster™ through the LRC Settings.

16. Advanced Pitch pRTI™ Network Performance Tuning

16.1. 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 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.

16.2. 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.

16.3. 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).

16.3.1. 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 the figure below.

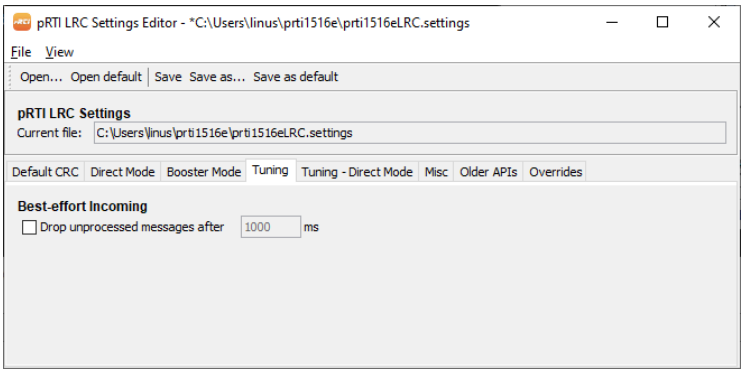

Figure 69. The general tuning settings in the LRC Settings GUI.

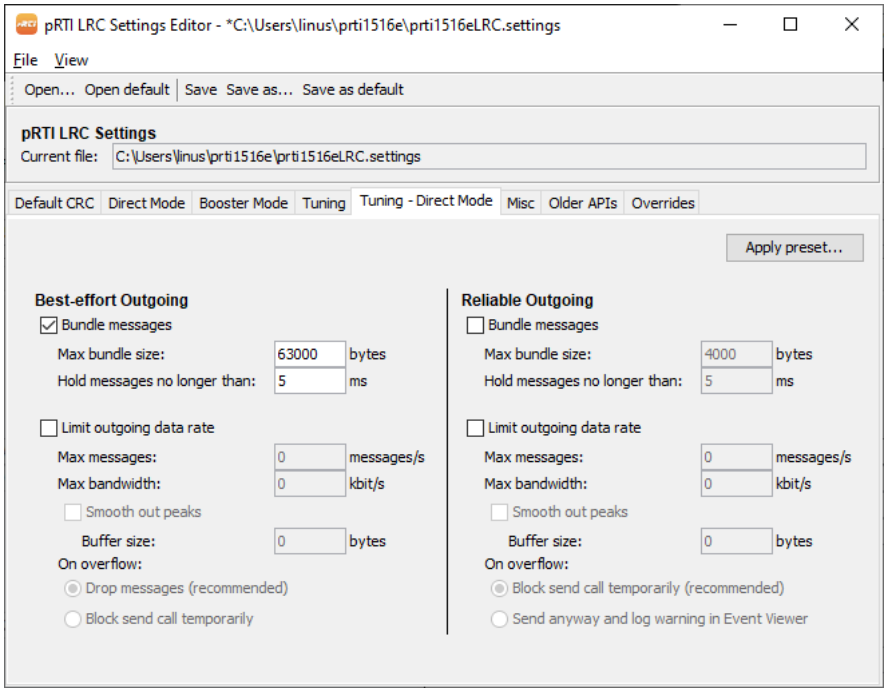

Figure 70. The tuning settings for direct mode in the LRC Settings GUI.

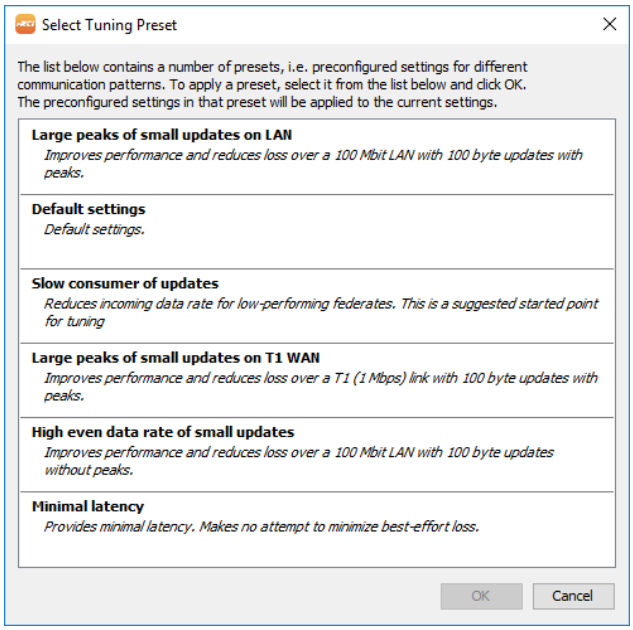

Figure 71. The tuning presets in the LRC Settings GUI.

To select a preset, click *Apply preset...* and select it in the tuning preset pop-up window. The predefined presets included with Pitch pRTI™ are listed in the table below.

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.

16.3.2. Advanced Tuning

Before you do any 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.

16.4. Pitch pRTI™ Tuning Algorithms

The figure below describes the chain of tuning algorithms used by Pitch pRTI™.

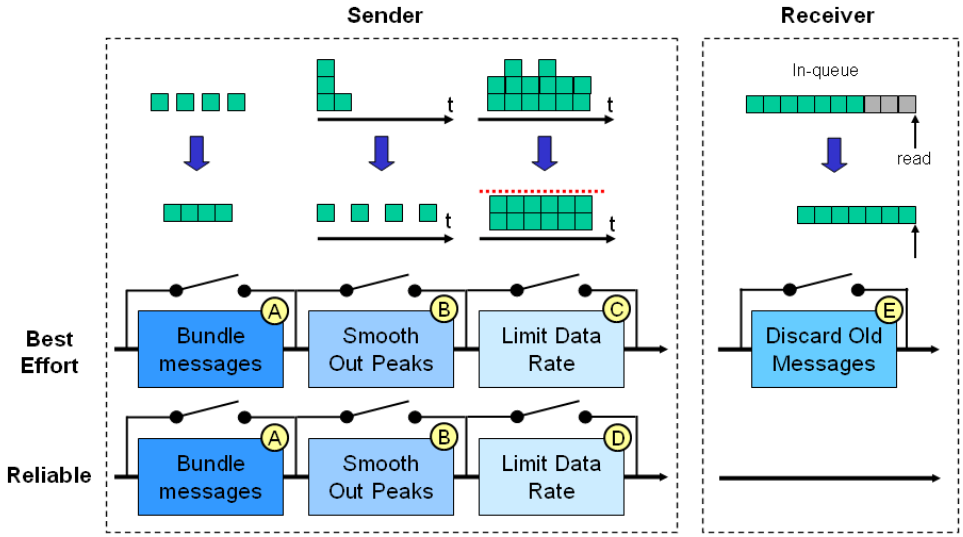

Figure 72. Pitch pRTI™ tuning algorithms.

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

16.4.1. 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.

The table below 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.

16.4.2. 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 16.4.3 and Section 16.4.4 for detailed descriptions of these two algorithms.

The table below 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.

16.4.3. 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.

16.4.4. 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.

16.4.5. 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.

16.4.6. 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.

The table below lists the parameters used by this algorithm.

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.

17. 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.

Settings are normally loaded from the `prti1516e` directory in your user home directory. If a CRC or LRC is started and this `prti1516e` directory does not exist, pRTI will create one containing factory default settings. As a reference, you can find a copy of these factory default settings under the `user.home` directory in the pRTI installation directory. If you want new users to use something other than the factory defaults, then you can use the `user.home` directory in the installation directory as a template to create your own custom settings. These custom settings must then be manually copied to the home directory of new users.

17.1. Settings for the Central RTI Component

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

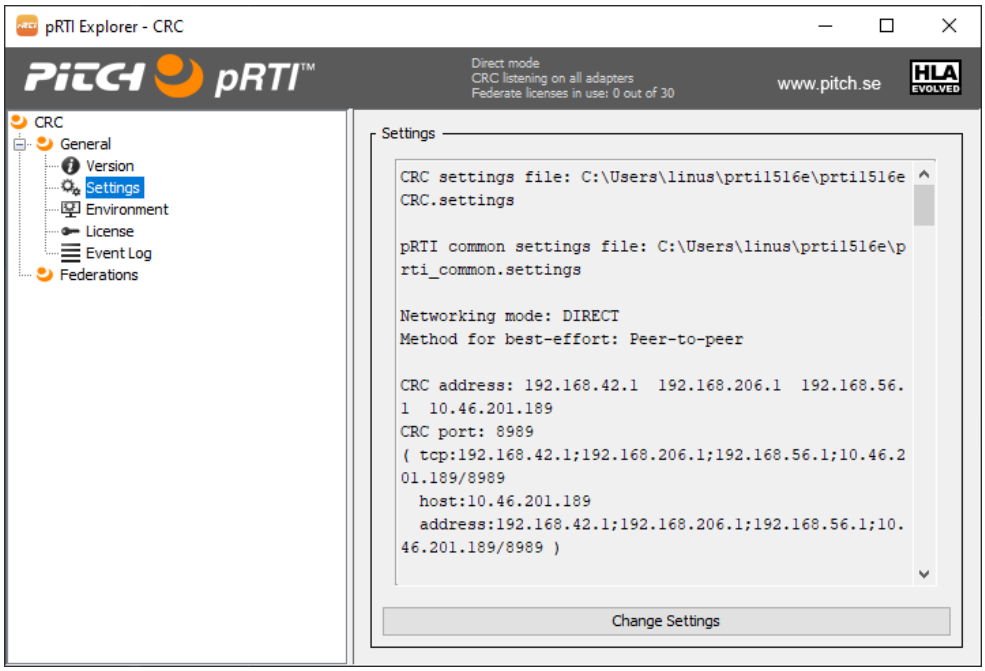

Figure 73. The CRC specific settings.

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. 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 CRCs on the same computer you can use the Java property `settings.dir` when starting the CRC (see example below). 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.

17.1.1. CRC Settings Editor

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

17.1.2. General Settings

The figure below shows the *General* settings tab in the *CRC Settings editor*.

Figure 74. The General tab in CRC settings editor.

The table below describes the settings available in the General tab for the CRC.

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.
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.
Object Instance Naming Strategy	This setting determines how the RTI generates object instance names when no name is provided by the federate. The first strategy creates names consisting of a prefix "HLA.", the final part of the class name and a serial number, for example "HLA.Car107". Names that start with the text "HLA" cannot be reserved by a federate which means that these objects cannot be recreated with the same names, for example when playing back a recording or when mirroring objects to another federation. The second strategy creates names that combine the final part of the class name, a serial number and a GUID (globally unique identifier). These names are reservable by a federate and can therefore be used for example when mirroring object instances between federations. The third strategy uses a custom name pattern provided in the settings. See description of name pattern below. It is critical that the names generated by this pattern are unique regarding both generated names and federate-provided names. Duplicate names can generate unexpected errors.

17.1.2.1. Name Pattern

The *Name Pattern* determines how pRTI generates instance names when no name is specified in the registerObjectInstance call.

The Name pattern can contain markers that are expanded by the RTI. Available markers are:

`%N` - full class name, including `HLAobjectRoot`

`%n` - last component of class name

`%m` - full class name, excluding `HLAobjectRoot`

`%i` - instance id

`%f` - federate id of registering federate

`%F` - name of registering federate

`%g` - guid

`%d` - date of registration

`%t` - time of registration

Example: with the Name pattern `%n_%i`, an instance of the RPR-FOM class `HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft` may get the instance name `Aircraft_215` where `215` is the numeric instance id.

Some name patterns can generate duplicate names. For example, the name pattern `N%` will generate the same name for all instances of the same class. If a federate tries to register an instance and the name pattern generates a name that is already in use, the `registerObjectInstance` service will throw an exception of type RTIinternalException with a descriptive error message, such as:

```
hla.rti1516e.exceptions.RTIinternalError: Name pattern 'ACD_duplicate_%n' for
object class 'HLAobjectRoot.A.C.D' generated duplicate name 'ACD_duplicate_D'
```

17.1.3. Network Settings

The figure below shows the *Network* settings tab in the *CRC Settings editor*.

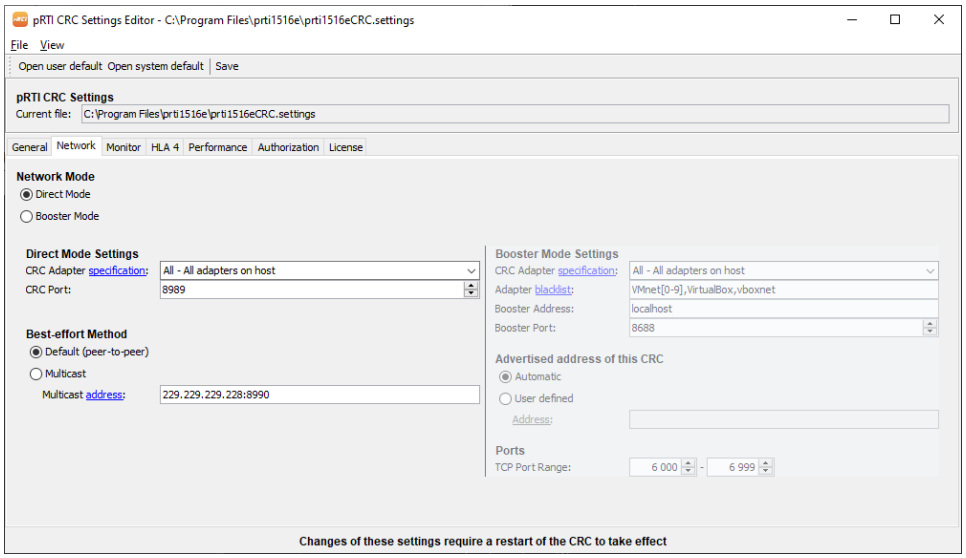

Figure 75. The Network tab in CRC settings editor.

The table below describes the settings available in the Network tab for the CRC.

Parameter name	Description
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	Specifies whether or not to use multicast to deliver best-effort messages.

Parameter name	Description
Multicast Address	The range of multicast addresses and ports. When a federation is created, the CRC picks a unique multicast address **and** a unique port in the specified range and assigns it to the federation. For example, the range 229.229.229.227-229.229.229.228:8990-8999, will yield two unique combinations, not 20. This also means that the range 229.229.229.228:8990-8999 yields one available combination since there is only one available address. The CRC keeps track of multicast addresses and ports in use to avoid collisions. Note that the CRC does not check if the address or port is in use by another application. Therefore, it is vital that the configured range is available for exclusive use by the CRC. A range uses the format `<address range>:<port range>`. If the address range is just a single address then it can be specified as `<address>`, otherwise it is `<first-address-in-range>-<last-address-in-range>`. If the port range is just a single port then it can be specified as `<port>`, otherwise it is `<first-port-in-range>-<last-port-in-range>`. Multiple ranges can be specified delimited by comma, for example: `239.0.0.1-239.0.0.2:3000-4000, 239.0.0.3:5001` The recommended address range for multicast is `239.0.0.0` to `239.255.255.255` and the recommended port range is `1024-32768`.
(Multicast Port)	(The port number entry is now part of the range setting. The CRC remains compatible with old settings files. If the Multicast address field does not contain a colon (denoting a range) then the address field is combined with this Multicast Port field to create a Multicast group address.)
Advertised address of this CRC	This setting can be used to enforce the advertising of a specific IP-address.
TCP Port Range	The CRC opens a port in this range for point-to-point connections with LRCs connected to the same Booster.

17.1.4. Monitor Settings

The figure below shows the *Monitor* settings tab in the *CRC Settings editor*.

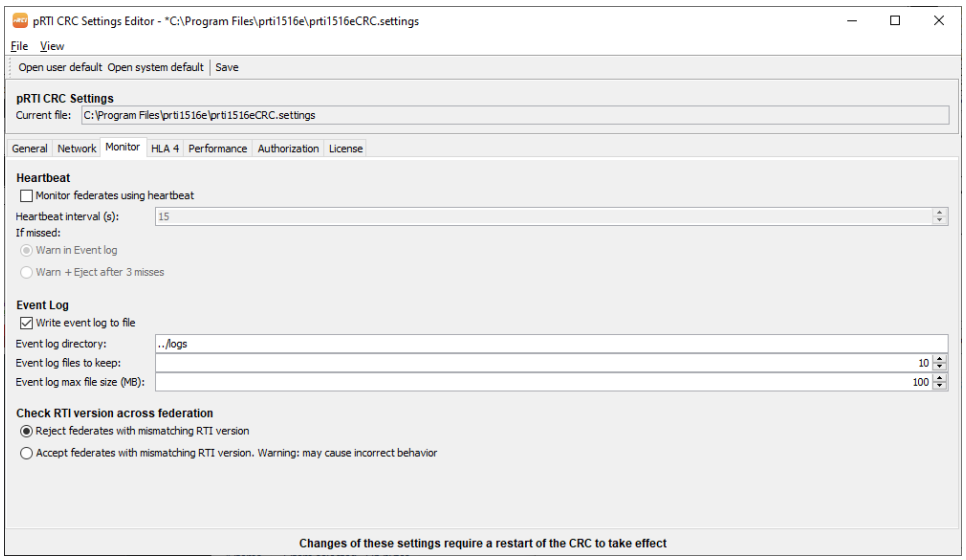

Figure 76. The Monitor tab in CRC settings editor.

The table below describes the settings available in the Monitor tab for the CRC.

Parameter name	Description
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

Parameter name	Description
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.

17.1.5. HLA 4 Settings

The figure below shows the *HLA 4* settings tab in the *CRC Settings editor*.

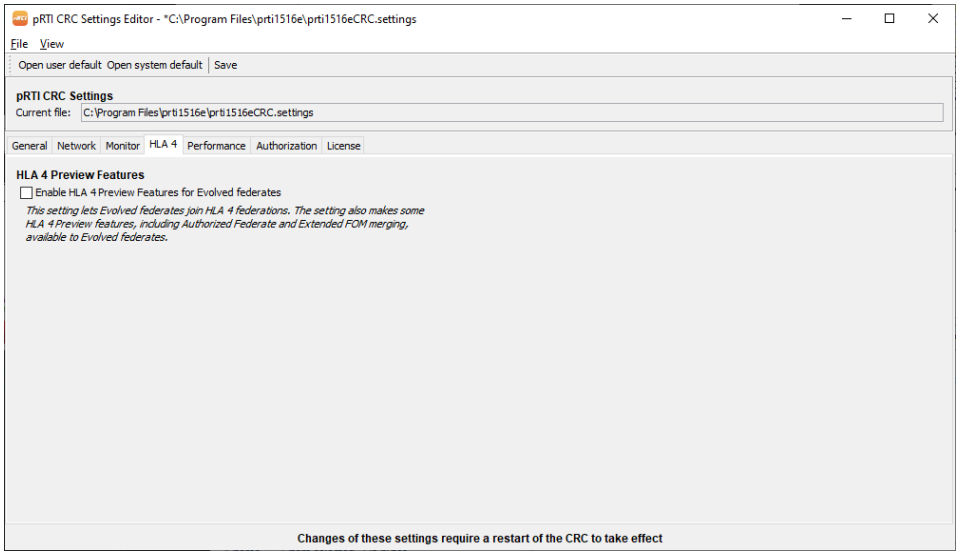

Figure 77. The HLA 4 tab in CRC Settings editor.

This option lets Evolved federates join HLA 4 federations. It will also enable some HLA 4 Preview features for Evolved federates, including Federate Authorization and Extended FOM merging. HLA 4 Preview is always available to HLA 4 federates. See Section 1.5 for more information on HLA 4 preview features.

17.1.6. Performance settings

The figure below shows the *Performance* settings tab in the *CRC Settings editor*.

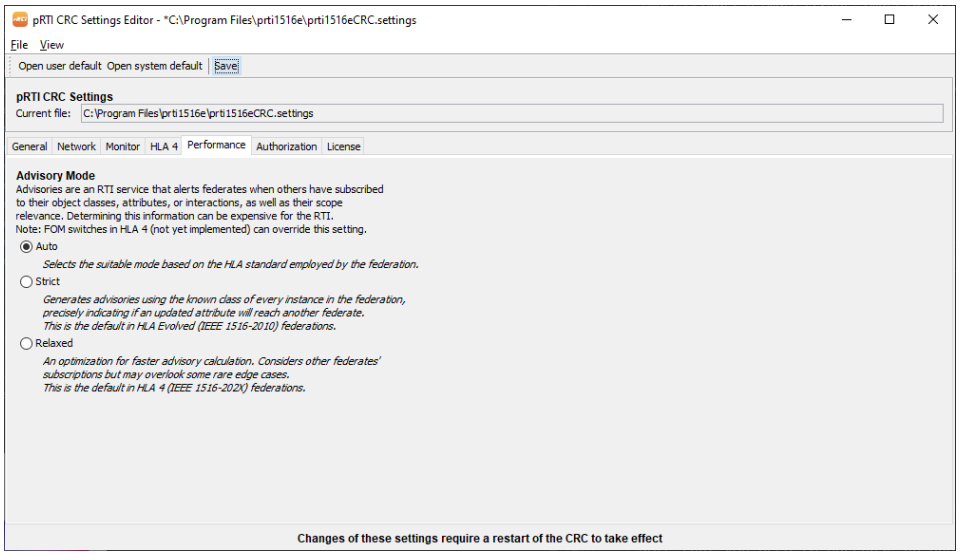

Figure 78. The Performance tab in CRC Settings editor.

The table below describes the settings available in the *Performance* tab for the *CRC Settings editor*.

Parameter name	Description
Auto	This option allows advisory mode to be set automatically depending on which HLA standard the federation is using.
Strict	Use Strict mode for all federations.
Relaxed	Use Relaxed mode for all federations.

See Section 1.5 for more information on Strict vs Relaxed Advisories.

17.1.7. Authorization settings

The figure below shows the *Authorization* settings tab in the *CRC Settings editor*.

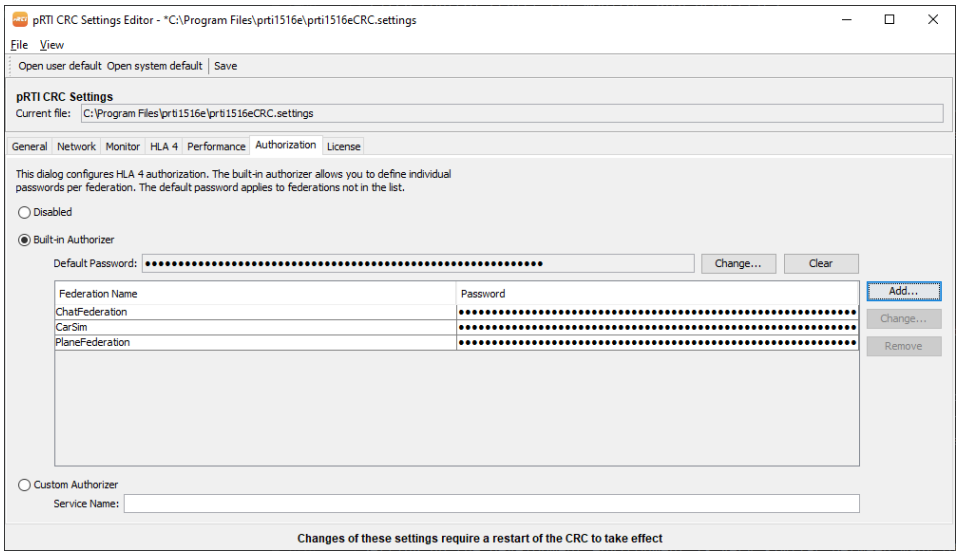

Figure 79. The Authorization tab in CRC Settings editor.

The table below describes the settings available in the *Authorization* tab for the *CRC Settings editor*.

Parameter name	Description
Disabled	Disables Authorization checks in the CRC.
Built-in Authorizer	Enables Authorization checks in the CRC using the built-in Authorization Service.
Default Password	Password that applies to federations that don't have a specific password.
Add/Change/Remove	Modify passwords for individual federations.
Custom Authorizer	Enables Authorization checks in the CRC using a custom Authorization Service.
Service Name	Specify the name of a custom Authorization Service.

See Section 1.5 for more information on Authorized Federates.

17.1.8. License Settings

The figure below shows the *License* settings tab in the *CRC Settings editor*.

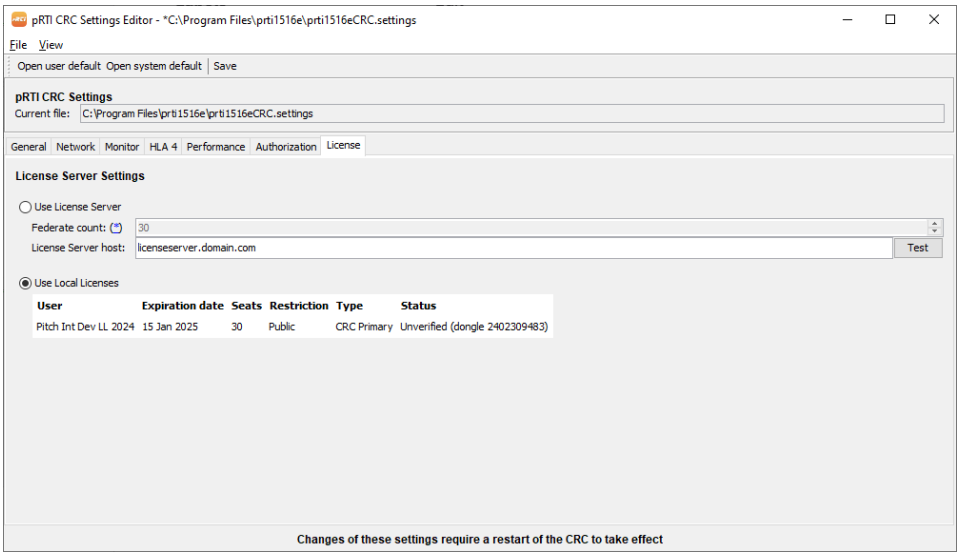

Figure 80. The License tab in CRC settings editor.

The table below describes the settings available in the License tab for the CRC.

Parameter name	Description
Use License Server	This option will set the pRTI to use a Pitch Floating License Server. Number of federates as well as the IP to the License Server should be specified.
Use Local Licenses	This option will set the pRTI to use available local licenses. These can be configured with the LicenseActivator and LicenseActivatorGUI applications.

17.2. 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 Section 15 and Section 16.

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.

17.2.1. 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 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.

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.

17.2.2. 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.

17.2.3. Default CRC

The figure below shows the *Default CRC* settings tab in the *LRC Settings editor* .

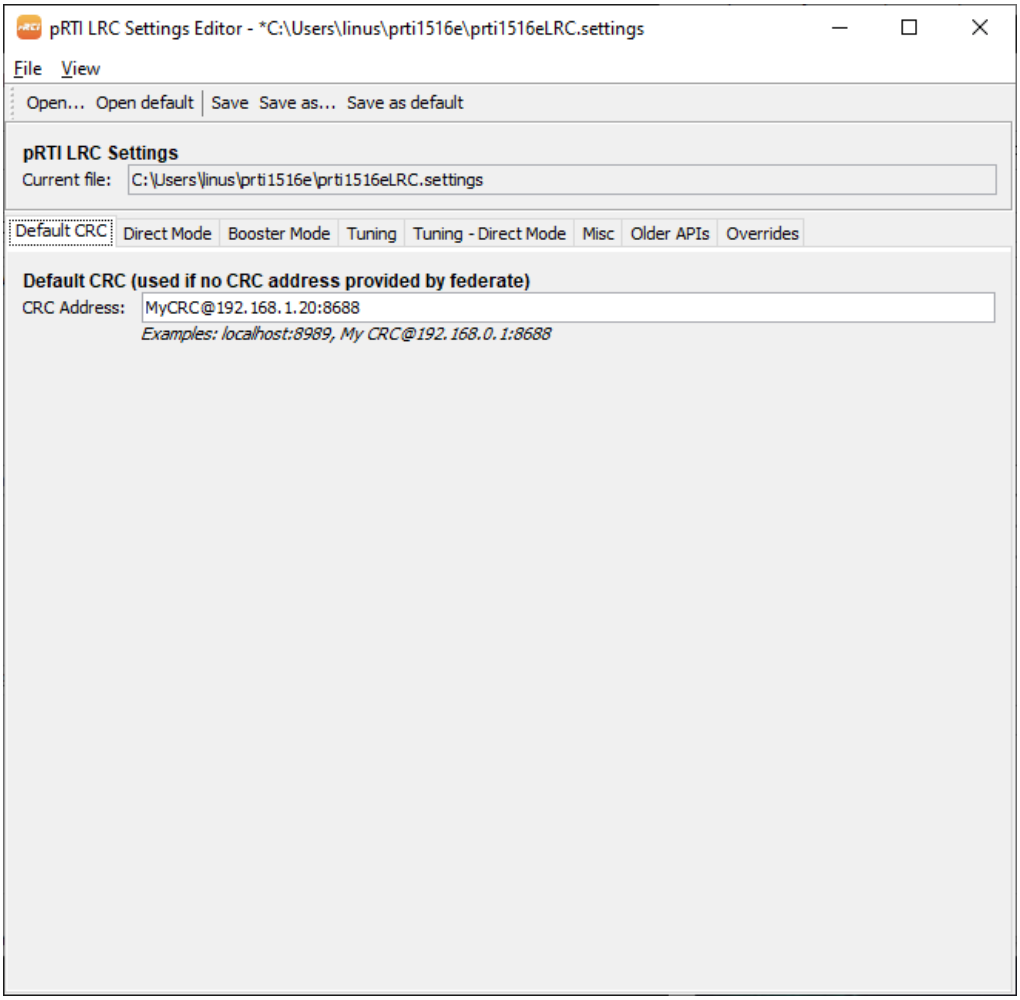

Figure 81. The Default CRC settings tab.

The table below describes the settings available in the *Default CRC* tab for the LRC.

General Parameters	Description
CRC-address	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. Depending on how the CRC-address is formatted, a direct network connection mode or a booster connection mode will be used. By setting the CRC-address to `hostname:port` or to `IP-address:port` you will use a regular, direct network connection which is the case when not using boosters at all. By setting the CRC-address to `CRC-name@booster-LAN-IP:port` you will use a connection through a booster network The button **[Find local CRC]** will automatically try to find a CRC on your local network for direct connections. The button **[Find Booster]** will be able to help you locate the local address to boosters on your network.

17.2.4. Direct Mode network Settings

The figure below shows the *Direct Mode* settings tab in the *LRC Settings editor*.

For a deeper discussion on network configuration see Section 15.

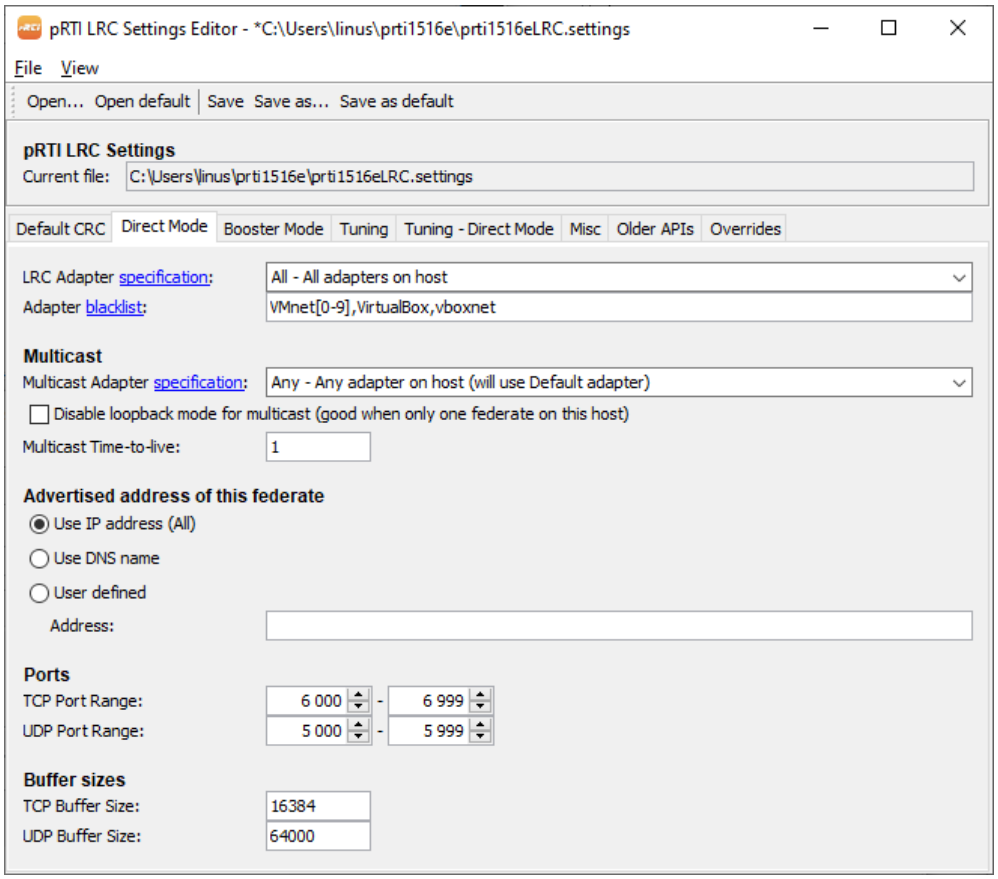

Figure 82. The LRC Settings Direct Mode tab.

The table below describes the settings available in the Direct Mode tab for the LRC.

Network Parameters	Description
LRC Adapter	Which IP address/addresses the LRC is using for network communication. Can be set to *All*, 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.

Network Parameters	Description
Multicast Adapter	Which network adapter to use when sending and receiving multicast traffic. Must be one and only one adapter. The *Any* setting will pick the *Default adapter* as defined by the Operating system. If the machine has more than one network adapter, it is strongly recommended to select the correct adapter here. Adapter specification can be an exact or partial match for adapter IP, adapter name, or adapter description. A partial match must be unique.
Advertised address of this federate	This setting can be used to enforce the advertising of a specific IP-address. When using the "Use IP address" option, a *black list* of IP-addresses, interface names or patters thereof can be defined to avoid advertising the address of certain interfaces.
TCP Port Range	The port range used by the LRC for connecting to either the CRC or different LRCs.
UDP Port Range	The same as *TCP Port Range* 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.

A note on buffer sizes and MTU

MTU stands for *Maximum Transfer Unit* which is the largest segment that can be sent on the network. The MTU is defined by the operating system and the network. The smallest MTU possible is 1500 bytes. Different network adapters can have different MTU. For example, Linux has recently increased MTU for the loopback adapter from 16384 bytes to 65536 bytes.

The receive buffer size for TCP and UDP must be larger than MTU. If the receive buffer is smaller than the MTU then there can be network packets larger than the receive buffer. These packets cannot be received and data will be lost. This is true both for UDP and TCP

17.2.5. Booster Mode network Settings

The figure below shows the *Booster Mode* settings tab in the *LRC Settings editor*.

For a deeper discussion on network configuration see Section 15.

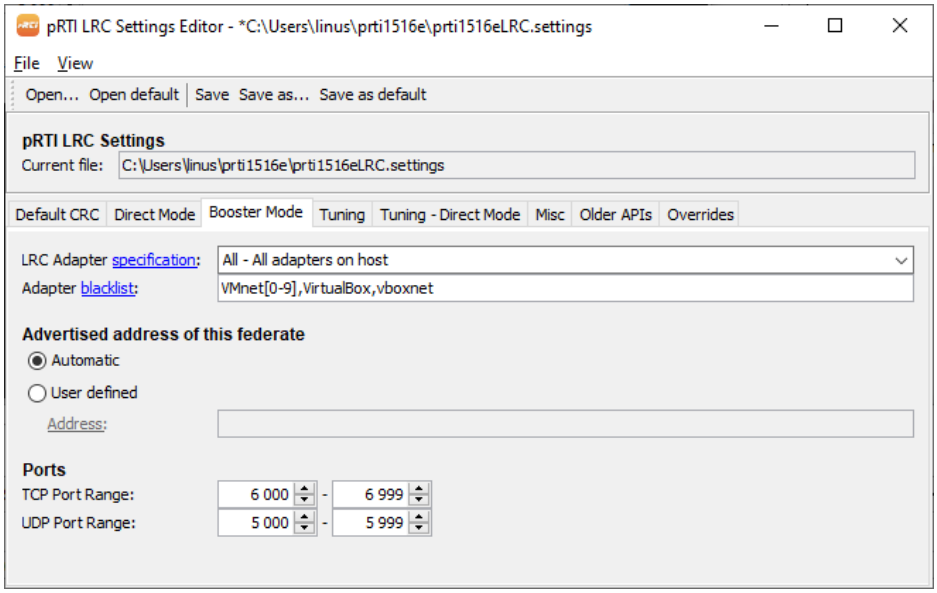

Figure 83. The LRC Settings Booster Mode tab.

The table below describes the settings available in the Booster Mode tab for the LRC.

Network Parameters	Description
LRC Adapter	Which IP address/addresses the LRC is using for network communication. Can be set to *All*, 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 blacklist	A comma-separated list of which adapters that pRTI should not use.
Advertised address	This setting can be used to enforce the advertising of a specific IP-address. When using the "Use IP address" option, a *black list* of IP-addresses, interface names or patters thereof can be defined to avoid advertising the address of certain interfaces.

Network Parameters	Description
TCP Port Range	The LRC opens a port in this range for point-to-point connections with other LRCs/CRC connected to the same Booster.
UDP Port Range	The same as *TCP Port Range* but for UDP.

17.2.6. Tuning Settings

The figure below shows the general *Tuning* settings tab in the *LRC Settings editor*. More tuning settings for Direct Mode can be done in the *Tuning - Direct Mode* tab.

For a deeper discussion on network tuning see Section 16.

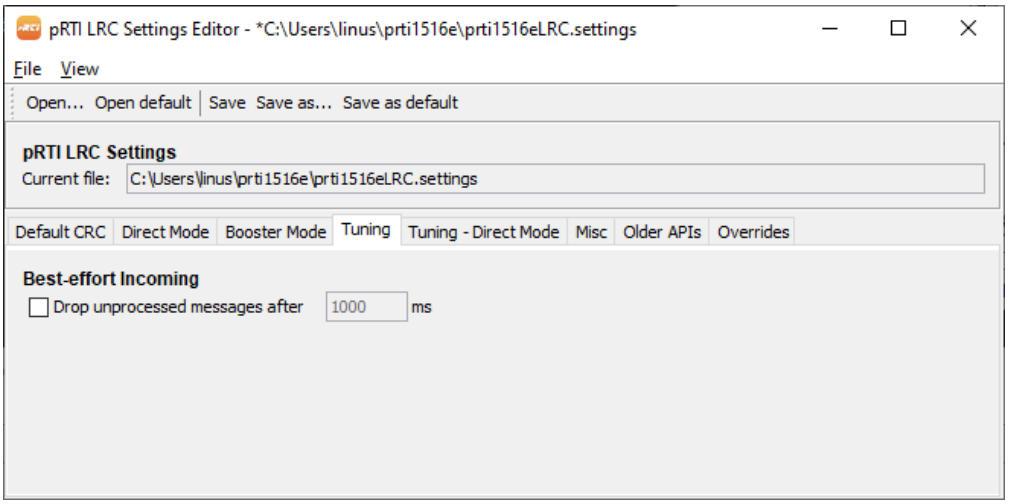

Figure 84. The LRC Settings Tuning tab.

The table below describes the settings available in the Tuning tab for the LRC.

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.

17.2.7. Tuning - Direct Mode Settings

The figure below shows the *Tuning - Direct Mode* settings tab in the *LRC Settings editor*.

For a deeper discussion on network tuning see Section 16.

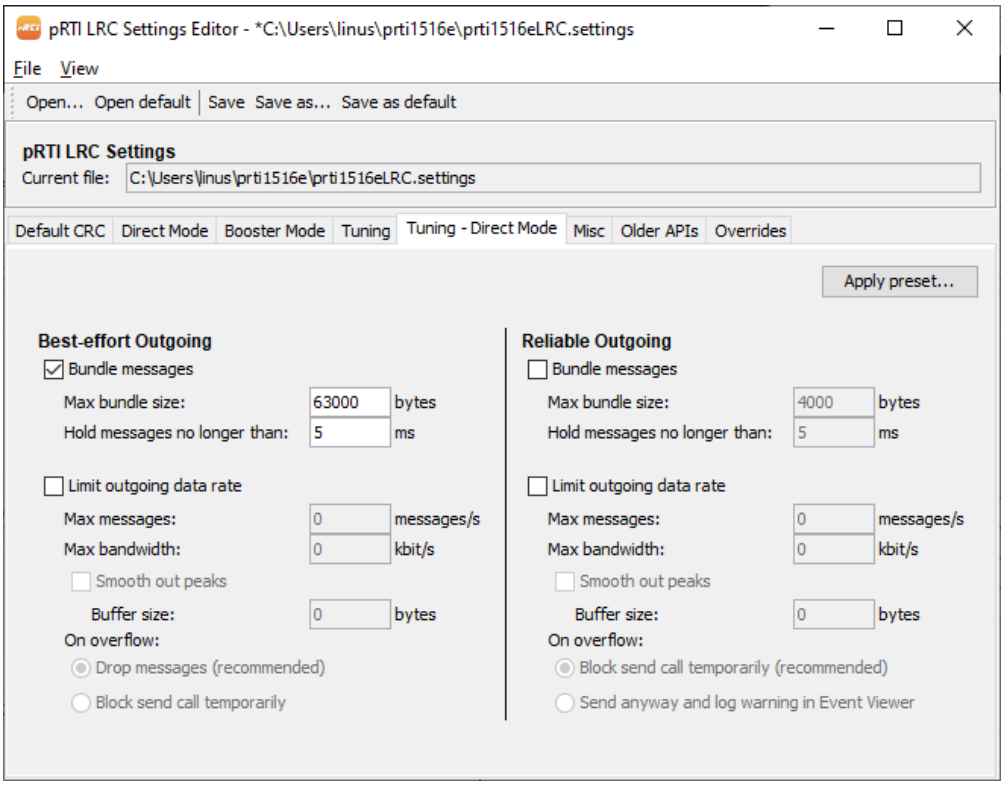

Figure 85. The LRC Settings Tuning - Direct Mode tab.

The tables below describe the settings available in the *Tuning - Direct Mode* tab for the LRC.

Best-effort Tuning Parameters	Description
Bundle messages	If enabled the RTI will bundle multiple messages before delivering. See Section 16.4.1 for details.
Max bundle size	The maximum size of a bundle when *Bundle messages* 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 then specified by the following two parameters. See Section 16.4.2 for more details.
Max. messages	The maximum number of messages to send per second when *Limit outgoing data rate* is checked.
Max bandwidth	The maximum send rate (in bits/s) produced by the RTI when *Limit outgoing data rate* is checked.
Smooth out peaks	Smooths out large peaks of data if *Limit outgoing data rate* is checked. See Section 16.4.3 for more details.

Best-effort Tuning Parameters	Description
Buffer size	The buffer size used when *Smooth out peaks* is checked.
On overflow	Specifies what to do when the buffer is full and *Limit outgoing data rate* 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 16.4.1 for details.
Max bundle size	The maximum size of a bundle when *Bundle messages* 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 then specified by the following two parameters. See section Section 16.4.2 for more details.
Max. messages	The maximum number of messages to send per second when *Limit outgoing data rate* is checked.
Max. bandwidth	The maximum send rate (in bits/s) produced by the RTI when *Limit outgoing data rate* is checked.
Smooth out peaks	Smooths out large peaks of data if *Limit outgoing data rate* is checked. See section Section 16.4.3 for more details.
Buffer size	The buffer size used when *Smooth out peaks* is checked.
On overflow	Specifies what to do when the buffer is full and *Limit outgoing data rate* is enabled.

17.2.8. Misc Settings

The figure below shows the *Misc* settings tab in the *LRC Settings editor*.

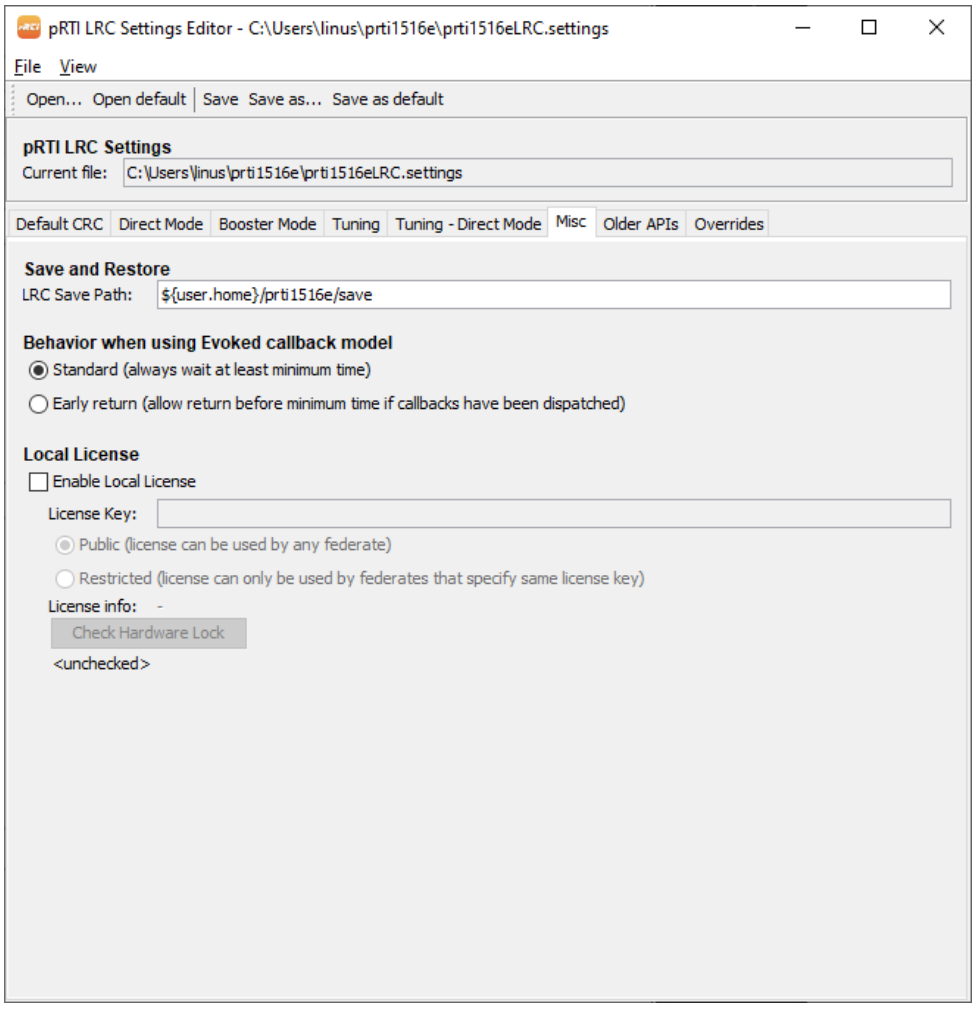

Figure 86. The LRC Settings Misc tab.

The table below describes the settings available in the *Misc* tab for the LRC.

Misc Parameters	Description
LRC Save Path	The path used for save and restore
Behavior 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 3.3.2 for further explanation)
License key	A license activation key for a local LRC license

Misc Parameters	Description
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

17.2.9. Older APIs settings

The figure below shows the *Older APIs* settings tab in the *LRC Settings editor*.

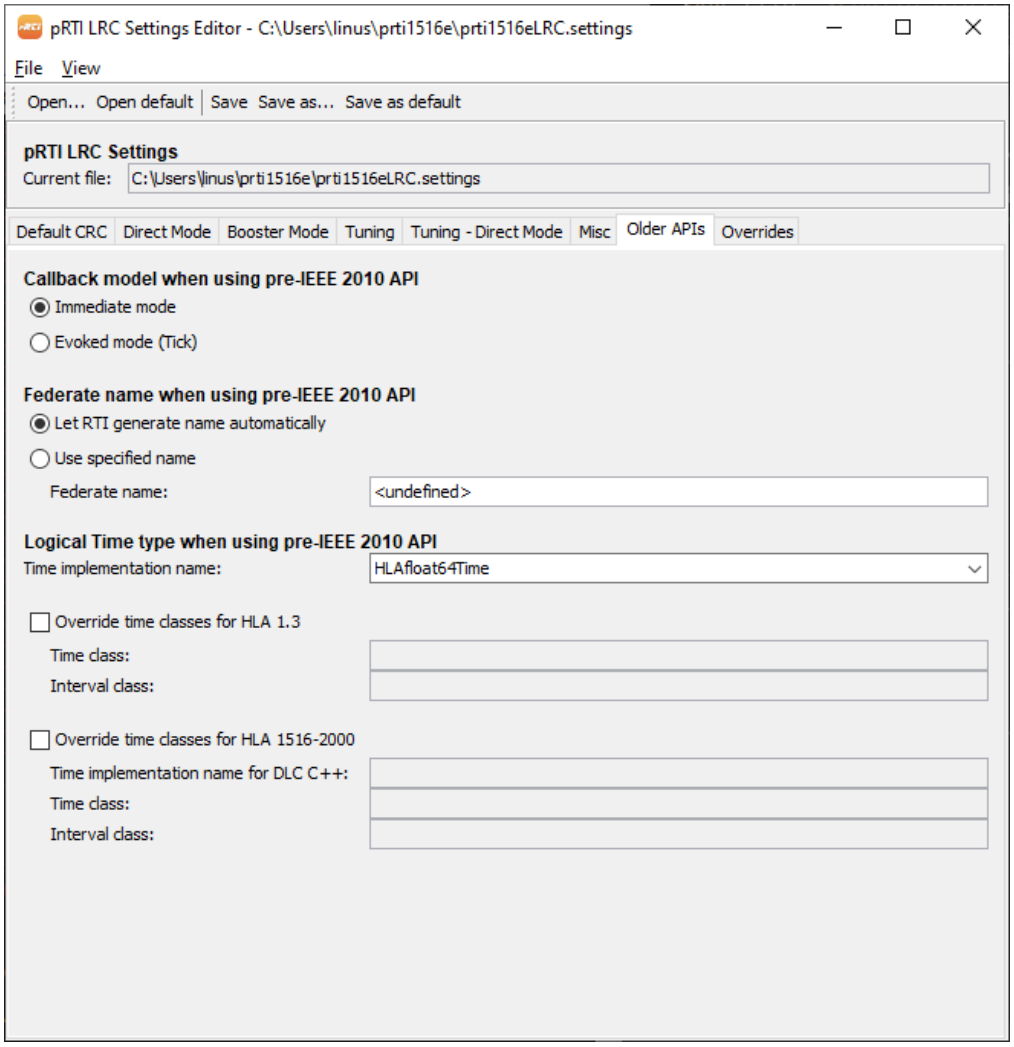

Figure 87. The LRC Settings Older APIs tab

The table below describes the settings available in the *Older APIs* tab for the LRC.

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"

Older APIs Parameters	Description
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

17.2.10. Override settings

The figure below shows the *Overrides* 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.

For more information about the password override, see Section 1.5.

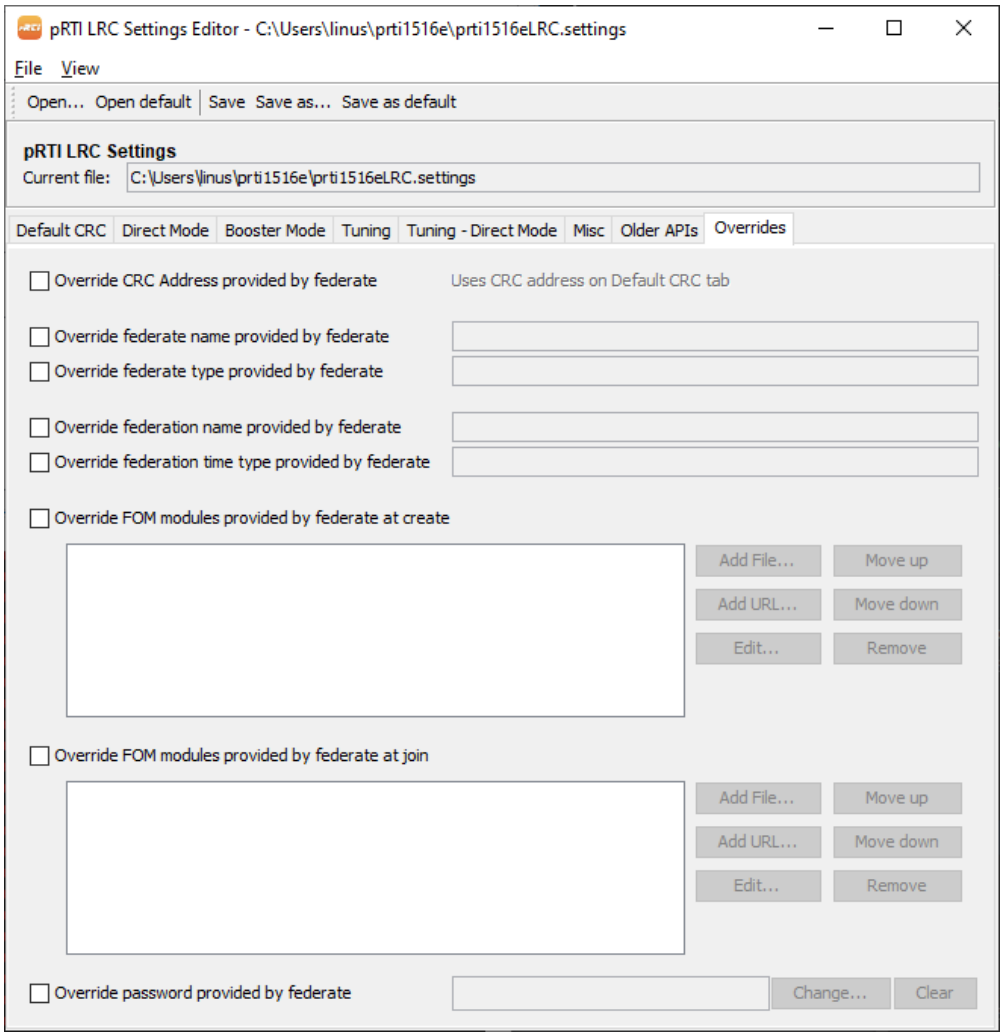

Figure 88. The LRC Settings Overrides tab

17.2.11. Additional overrides settings

There are some override settings which are currently not available in the settings GUI but can be set using a text editor or the text mode of the LRC Settings application (View → Text mode).

Setting	Description
LRC.namePattern	A pattern that determines how pRTI generates instance names when no name is specified in the registerObjectInstance call. Note that these settings will override the Naming Strategy setting in the CRC settings. The Name pattern can contain markers that are expanded by the RTI. These markers are described in Section 17.1.2.1. Patterns for individual classes can be specified by appending sections of the class name to the setting name. For example, all RPR-FOM platforms can use a specific name pattern by providing the setting: `LRC.namePattern.HLAobjectRoot.BaseEntity.PhysicalEntity.Platform=RPR_platform_%n_%i` Class-specific name patterns can be specified for different levels in the class hierarchy. The most specific match will be used for name pattern. Some name patterns can generate duplicate names. Duplicate names can generate unexpected errors.

17.3. 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 256 megabytes) 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.

17.4. Troubleshooting LRC & CRC Settings

Sometimes it's not obvious where a setting comes from and what value it has. This is especially complicated when running a CRC in service mode where there is no user interface to check or when an application fails to start.

To troubleshoot settings, add this setting to the appropriate `.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):

```
-Dse.pitch.prti1516e.traceSettings=true
```

This will make the CRC or LRC log the value and source of every setting to stdout.

For a CRC running in service mode, this output can be found in the file `CRC_service_stdout.log` in the `logs` subdirectory of the pRTI installation (Typically `C:\Program Files\pRTI1516e` on Windows and `/usr/local/prti1516e` on Linux.)

For a CRC running in user mode without a console, this output can be found in the file `CRCstdout.log` in the same location.

17.4.1. CRC Settings Load Order

Here's how CRC loads settings.

1. Check for user-defined settings
   a. If user has specified command line argument `-settings <file>` then load that file
   b. Else if user has set system property `settings.file` then load that file.
   c. Else if user has set system property `settings.dir` then load `<settings.dir>/prti1516eCRC.settings`.
2. If user did not specify user-defined settings
   a. If service mode
   Load `prti1516eCRC.settings` from system-wide dir (`PRTI1516E_HOME` on Windows, `/etc/prti1516e` on Linux)
   b. Else (not service mode)
   if default settings dir `user.home/prti1516e` doesn't exist, copy default settings dir to `user.home/prti1516e`
   Load `prti1516eCRC.settings` from default settings dir
3. Finally, override loaded settings with system properties

17.4.2. LRC Settings Load Order

Here's how LRC loads settings.

1. Check for user-defined settings
   a. If user has set system property `settings.file` then load that file.
   b. Else if user has set system property `settings.dir` then load `<settings.dir>/prti1516eLRC.settings`.
2. If user did not specify user-defined settings
   a. if default settings dir `user.home/prti1516e` doesn't exist, copy default settings dir to `user.home/prti1516e`
   b. Load `prti1516eLRC.settings` from default settings dir
3. Finally, override loaded settings with system properties

Then, LRC processes LSD (Local Settings Designator) in the following steps.

1. Check for replacement
   a. if current directory contains a file `alternate-lsd.txt`. If it does, read lsd from that file.
   b. if environment variable `alternate_lsd` exists, use its value as lsd.
   c. if system property `alternate.lsd` exists, use its value as lsd.
2. If lsd is empty, replace with abstract reference `default`.
3. Read settings from file
   a. if settings designator is abstract reference
      i. read settings from file `prti.home/settings/<abstract reference>.lsd`, if it exists
      ii. read settings from file `user.home/settings/<abstract reference>.lsd`, if it exists
      iii. read settings from file `currentDir/<abstract reference>.lsd`, if it exists
   b. else if settings designator is file designator, read settings from that file
   c. else if settings designator is URL designator, read settings from that URL
4. Override accumulated settings with settings from lsd.

18. Using pRTI in Containers

This section documents port usage by pRTI components. This information is needed if you wish to run pRTI in containers such as Docker or Vagrant. This section assumes that you are familiar with containers, port forwarding, etc. It is not a tutorial on containers.

Pitch Booster in Bridged mode simplifies deployment in containers. No additional configuration will be necessary if Pitch Booster is used.

Port forwarding, as described here, is required when the container is using network isolation. Depending on container technology used, containers can usually also be configured in other network modes that expose the internal ports, for example Docker bridged mode or Vagrant public networks, where port forwarding is not required.

18.1. Component overview

A pRTI deployment consists of a number of components, as shown in this figure:

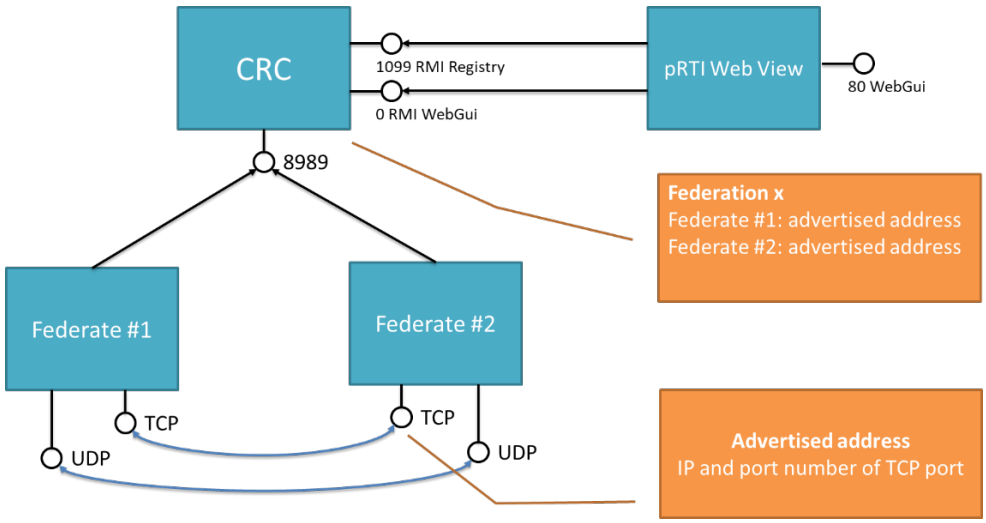

Figure 89. pRTI Deployment Overview.

The CRC opens one port (default: 8989) where it listens for connections from federates. It also opens an RMI Registry port (default: 1099) that is used by pRTI Web View. Finally, it opens a WebGui service port that is used by the pRTI Web View service. This port is by default *anonymous*, which means that the operating system automatically selects an available port.

Each federate opens one TCP port and one UDP port, selecting an available port from the appropriate ranges. The federate has an advertised address that is listed in the CRC from which it is retrieved and used by other federates to connect to the original federate.

18.2. Running CRC in Container

This figure shows the CRC running in a container with the ports that need to be forwarded.

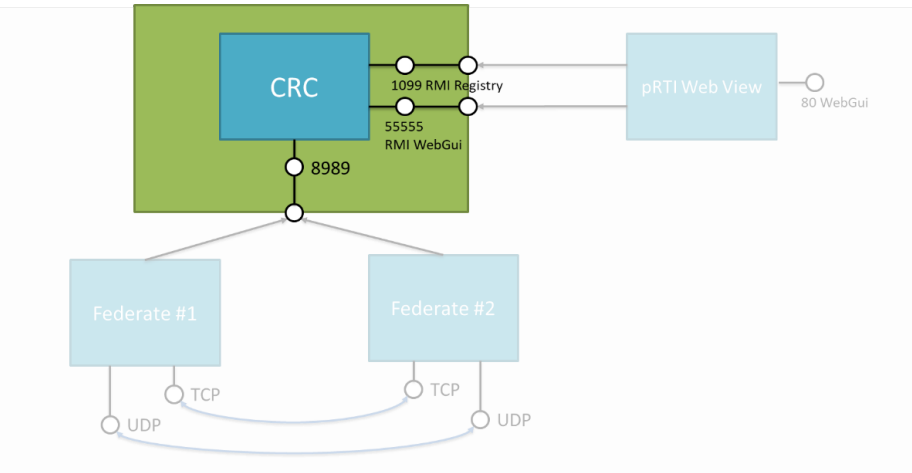

Figure 90. CRC in a Container.

These are the steps required to run pRTI CRC in a container:

Expose pRTI CRC port 8989

The pRTI CRC listens for connections on port 8989. This is the port where federates connect to the CRC. You must add port forwarding for this port in a way appropriate for your selected container type.

Disable multicast

Multicast is not always supported by containers, so another recommended change is to set pRTI Best-effort Method to peer-to-peer. In `prti1516eCRC.settings`, set:

```
Multicast.enabled=false
```

Configure Licensing

The recommended license type when running pRTI CRC in a container is to use Pitch Floating License Server. These are the required changes in `prti1516eCRC.settings`:

```
CRC.licenseType=server
CRC.license-server.host=<floating-license-server-host>
CRC.license-server.federateCount=<federate-count>
```

Licenses that are node-locked or use a USB dongle are not recommended for containers.

Expose Pitch Web View port

You can choose to run Pitch Web View Server in the same container as pRTI CRC or in a separate container.

If Pitch Web View Server runs in the same container, add port forwarding for port 80.

Running Pitch Web View Server in a separate container requires a bit more configuration. By default, pRTI CRC opens port 1099 plus an anonymous port to accept connections from Pitch Web View Server. Since it is not possible to forward an anonymous port, pRTI CRC must be configured to use a specific port. To accomplish this, pick a port number, e.g. 55555, and add the following setting to `prti1516eCRC.settings`:

```
prti1516e.remoteGui.port=55555
```

Now, pRTI will use ports 1099 and 55555 for Pitch Web View Server. These ports must be forwarded.

18.3. Running federate/LRC in Container

This figure shows a federate running in a container with the ports that need to be forwarded.

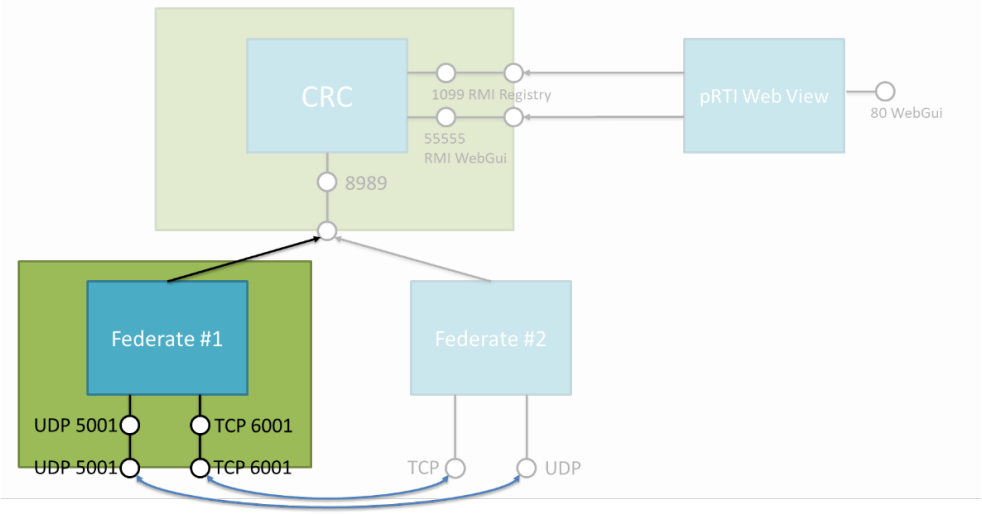

Figure 91. LRC in a Container.

A pRTI LRC accepts incoming traffic using one TCP port and one UDP port. The first step is to control which ports the federate uses.

Note that the guest and host ports must match. In this example, we will run one federate in a container. Federate 1 will use port 6001 for TCP and port 5001 for UDP.

In federate LRC settings:

```
LRC.TCP.port-range.start=6001
LRC.TCP.port-range.end=6001
LRC.UDP.port-range.start=5001
LRC.UDP.port-range.end=5001
```

Then, configure port forwarding of TCP port 6001 and UDP port 5001 for the container.

By default, a federate advertises its local address which in this case is the internal guest IP address. This address is not reachable from outside the container. Federate 1 must advertise the host address where other federates can reach federate 1 through port forwarding.

In federate 1 LRC settings, replacing <host-ip> with the IP address of the container host:

```
LRC.TCP.advertise.mode=User
LRC.TCP.advertise.address=<host-ip>
```

Repeat the same process for federate 2 running in another container on the same host. Federate 2 will use port 6002 for TCP and port 5002 for UDP.

In federate 2 LRC settings, replacing <host-ip> with the IP address of the container host:

```
LRC.TCP.port-range.start=6002
LRC.TCP.port-range.end=6002
LRC.UDP.port-range.start=5002
LRC.UDP.port-range.end=5002
LRC.TCP.advertise.mode=User
LRC.TCP.advertise.address=<host-ip>
```

Then, configure port forwarding of TCP port 6002 and UDP port 5002 for the container.

Finally, make sure that the Best-effort Method is Default (peer-to-peer) and not using multicast. Please refer to the section on running CRC in a virtual machine.

18.4. Containerization with Docker

A sample of how to make *Docker* containers with Pitch pRTI™ is included in the `samples/docker` directory. The sample can be used directly to build and run a container, or used as a starting point for building a customized container. The container can be used together with other technologies like *Docker Compose* and *Kubernetes*.

The following images are available:

- **CRC with WebView** in `samples/docker/Dockerfile`.
- **Federate/LRC base image** in `samples/docker/prti_lrc.dockerfile`.
- **C++ federate sample — chat-cpp-1516e** in `samples/chat-cpp-1516e/Dockerfile`.
- **Java federate sample — chat-java-1516e** in `samples/chat-java-1516e/Dockerfile`.

18.4.1. CRC with WebView image

This image can be used to run Pitch pRTI CRC and Pitch WebView Server in the same container.

Use the `Dockerfile` as the starting point to understand the sample. The CRC and WebView are started via the `docker-entrypoint.sh` script, which applies the necessary runtime configuration options before launching the application.

The `prti1516eCRC.setting` file is used for configuration. Some fields in the configuration file are modified by the start script to apply runtime options.

The `install-webview.sh` script is used when the image is built to install WebView. If the image is built from a Windows or Mac OS installation, the Linux WebView installer or `webview.war` has to be downloaded separately and placed in the `webviewinstaller64/` directory in Pitch pRTI™ install dir.

18.4.1.1. Build and run the container

From the Pitch pRTI™ install dir, run the following command to build the image:

```
docker build -t prti_crc -f ./samples/docker/Dockerfile .
```

To run the CRC and WebView, use the following command:

```
docker run -p 8989:8989 -p 8080:8080 prti_crc
```

TCP port `8989` is used for the CRC communication. TCP port `8080` is used for the WebView server.

Environment variables can be set directly when executing the run command with the `-e` argument, for example:

```
-e LICENSE_SERVER=<licence_server_URL>
```

The following environment variables can be configured for the container:

- `LICENSE_SERVER`, URL for Pitch Floating License Server. Default is `pfls`.
- `FEDERATE_COUNT`, Number of federate licenses to request from the License Server. Default is `5`.
- `DISABLE_WEB_VIEW`, Set to a non-empty string to disable WebView. Default is *not set*.
- `JAVA_OPTS`, Java runtime options. Default is `-XX:+UseParallelGC -XX:MaxRAMPercentage=75`.

18.4.2. Federate/LRC base image

This image can be used as a base image for C++ and Java-based federates using Pitch pRTI™.

Use the `prti_lrc.dockerfile` as the starting point to understand the sample.

The `prti1516eLRC.setting` file is used for configuration. Some fields in the configuration file can be modified by a start script to apply runtime options.

The `set-LRC-settings.sh` start script can be used to apply runtime options. See Section 18.3 for a description of the options.

The environment variables `PRTI1516_HOME`, `CLASSPATH`, and `LD_LIBRARY_PATH` are configured for C++ and Java federates.

18.4.2.1. Build the base image

From the Pitch pRTI™ install dir, run the following command:

```
docker build -t prti_lrc -f ./samples/docker/prti_lrc.dockerfile .
```

This image can now be used for C++ and Java based federates.

18.4.3. C++ federate sample — Chat C++

Use the `samples/chat-cpp-1516e/Dockerfile` as the starting point to understand the sample. This sample assumes that a Linux installation is used.

To build the image, run the following command from `samples/chat-cpp-1516e/` directory:

```
docker build -t chat-cpp-1516e .
```

If you have a container registry, that can be specified with the `PRIVATE_REGISTRY` argument:

```
--build-arg="PRIVATE_REGISTRY=gitlab.mycompany.com:5050/containers/"
```

Run the following command to start the Chat C++ sample. The address of the container host should be used as the value for `ADVERTISE_ADDRESS`.

```
docker run -p 6012:6012 -p 6012:6012/udp -e ADVERTISE_ADDRESS=1.2.3.4 -it
chat-cpp-1516e
```

TCP and UDP ports `6012` are used for the RTI LRC communication. The `ADVERTISE_ADDRESS` is used for the other federates to connect back to the Chat C++ federate.

18.4.4. Java federate sample — Chat Java

Use the `samples/chat-java-1516e/Dockerfile` as the starting point to understand the sample.

To build the image, run the following command from `samples/chat-java-1516e/` directory:

```
docker build -t chat-java-1516e .
```

If you have a container registry, that can be specified with the `PRIVATE_REGISTRY` argument:

```
--build-arg="PRIVATE_REGISTRY=gitlab.mycompany.com:5050/containers/"
```

Run the following command to start the Chat Java sample. The address of the container host should be used as the value for `ADVERTISE_ADDRESS`.

```
docker run -p 6010:6010 -p 6010:6010/udp -e ADVERTISE_ADDRESS=1.2.3.4 -it
chat-java-1516e
```

TCP and UDP ports `6010` are used for the RTI LRC communication. The `ADVERTISE_ADDRESS` is used for the other federates to connect back to the Chat Java federate.

18.4.5. Docker networking modes

Docker supports different networking modes.

The best networking mode to use depends on how the CRC or federate should connect to other federates, if they are running in containers, and if Docker Compose or Kubernetes are used.

Using a custom network, `docker network …`, for the containers should also be considered.

19. 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 17.2.

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

19.1. 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.

19.2. Headers

The `api` subdirectory of the installation contains API headers from each HLA version, including legacy headers. The headers are organised by HLA version in subdirectories of the installation directory. The name of each subdirectory reflects the HLA version of the API headers that are located in it. The subdirectories are:

- `api/cpp/HLA_1516-202X` (contains draft HLA 4 headers)
- `api/cpp/HLA_1516-2010` (contains HLA Evolved headers)
- `api/cpp/HLA_1516-2000`
- `api/cpp/HLA_1516-DLC`
- `api/cpp/HLA_1.3`

19.3. 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 is required to be on the CLASSPATH when running C++ federates using older HLA versions (i.e. HLA 1.3, IEEE 1516-2000 and IEEE 1516-2000 DLC) or when a JVM is instantiated outside of pRTI™.

19.4. 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.

19.4.1. 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.

19.4.2. 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`.

19.5. 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.

19.5.1. 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.

19.5.2. 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
```

19.6. 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.

20. Common errors

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

20.1. 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 10.1.

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 10.1.

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 10.1 for more information.

20.2. 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.rti1516e.*;

^
```

Reason: The compiler cannot find the `prti1516e.jar` file. Make sure you have the file available on your CLASSPATH. See Section 10.6.2 for more information.

20.3. 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 10.6.2 for more information.

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

```
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 17.1 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 LRCs.

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 message:*

```
Can't connect to RTIexec host. (909067003)

hla.rti1516e.RTIinternalError: CRC version does not support all required HLA 4
capabilities. (909067003)
```

Reason: The federate expects capabilities not present on the RTI. This is due to the CRC being older than the local LRC library used in the federate. Only applicable for HLA 4 federates.

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.

20.4. 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 Section 19.1 for more information.

20.5. 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, be aware that it contains a built-in firewall. If you are experiencing this problem when running Windows then check if the firewall is enabled. You can reach the firewall settings by first typing the `Windows key` + `S` then typing `firewall` and press `Enter`.

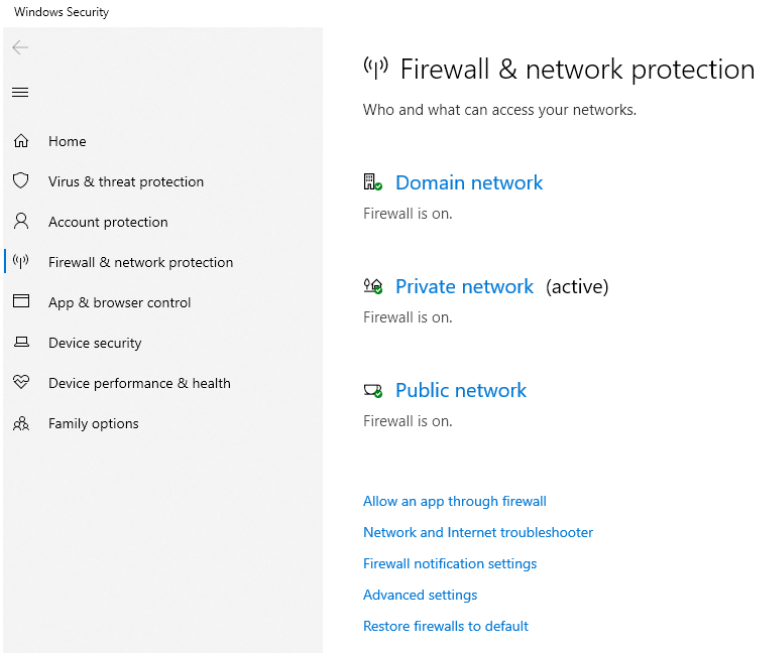

Figure 92. Disabling Internet Connection Firewall on Windows XP.

To disable the firewall, click on the appropriate network (Domain or Private) to access a settings dialog from where you can disable the firewall.

20.6. 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.

20.7. 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.

20.8. 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.

20.9. 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:*

```
// Problematic code
_rtiAmbassador.reserveObjectInstanceName(name);
synchronized (_reservationSemaphore) {
   try {
      _reservationSemaphore.wait();
   } catch (InterruptedException e) {
   }
}
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 multithreaded programming. What happens is that the callback is delivered before the caller has synchronized the semaphore. Instead, the callback synchronizes the semaphore and calls notify. After the callback has terminated, the caller continues and 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.

```
synchronized (_reservationSemaphore) {
   try {
      _rtiAmbassador.reserveObjectInstanceName(name);
      _reservationSemaphore.wait();
   } catch (InterruptedException e) {
   }
}
System.out.println("Reservation completed");
```

Problem: *I am experiencing packet loss, missing callbacks and unstable 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 Section 15 on how to configure the Pitch pRTI™ when operating with firewalls.

Pitch Technologies

Repslagaregatan 25, S-582 22 Linköping, Sweden

Tel: +46 (0)13 470 55 00

Web: https://www.pitchtechnologies.com, Email: info@pitchtechnologies.com