# Contents

1

# Preface

## Audience

This guide is designed to assist developers working on the .Net technology to understand and use FioranoMQ C#RTL library. The developer must have a fair knowledge of the C# programming language.

## Overview of the Guide

This guide contains the following contents.

- An introduction to FioranoMQ C# RunTime library.
- Detailed description of the APIs included in this library.
- Details on compiling and running C#RTL sample applications.
- Frequently Asked Questions (FAQ).

## Introduction to FMQ C#RTL

The C#RTL allows C# applications to communicate seamlessly with C, C++ and java programs. This library helps in bridging the gap between J2EE and .NET domain and enables .NET applications to directly talk to the Java server. This version of C#RTL supports secure and non-secure TCP and HTTP connections on Win32 platform for Point-to-Point and Publish/Subscribe communication models.

The C#RTL has been designed to provide maximum conformance to JMS specifications. All public APIs have similar signature as the corresponding Java APIs specified by JMS. All classes have a similar naming convention. Exception handling is also similar to that specified by JMS with all APIs throwing a CJMSException with a specific error Code and/or error description.

---

☞ For more information on JMS specifications, please refer to Java Message Service Specification on the Sun Microsystems website.

---

# Typographical Conventions

To locate and interpret information easily, the manual uses consistent visual cues, and standard text formats. The Table below lists the various conventions used in the manual.

| Conventions | Description |
|---|---|
|  Note | Additional/important information. |
|  See Also | Reference to important information located elsewhere in the documentation framework. |
|  Warning | Information regarding a potential aberration if a component is used incorrectly. |
| 1. Numbered List | Instructions that must be followed in a sequential order. |
| ■ Bulleted List | A list where sequence is immaterial. |
| C:\Programs\FioranoMQ | Directory paths |
| start() | Class names, methods, and interfaces. |
| `public class ChatService extends FioranoBISService` | Code samples |
| `fiorano.FioranoBIS.common.*` | This style is used to highlight:<br>1. Program code within paragraphs<br>2. Values to be entered by users in a procedure |

**TABLE 1 Typographical Conventions**

# Related Documentation

For complete information on the available RunTime libraries, we strongly recommend that you go through the entire range of FioranoMQ RTL Documentation.

| Document Name | Description |
|---|---|
| C Runtime Library Guide | Contains the description of the FioranoMQ C Runtime APIs. |
| Native C ++ Runtime Library Guide | Contains a description of the FioranoMQ native C++ Runtime APIs. |

| Document Name | Description |
| --- | --- |
| JNI-based C ++ Runtime Library Guide | Provides an introduction toFioranoMQ JNI based C++ Runtime APIs |
| JavaDocs | Contains "javadoc" style documentation on all public Fiorano classes. |

**TABLE 2 Related Documentation**

2

# Datatypes and Constants

This chapter contains an overview of the CRTL specific data types and constants. A brief explanation of each data type along with sizes, is provided.

## Naming convention

The C#RTL adheres to the following naming convention for you to easily identify the classes, constants and member functions with the corresponding definitions in JMS specifications.

| Type | Naming Convention | Example |
|---|---|---|
| Class | Cs<JMSClass name> | `CsTopicPublisher(JMS:Top-icPublisher)` |
| Constant | same as in JMS spec | `NON_PERSISTENT` |
| Function | same as in JMS spec | `publish` |

# Error Handling

3

The C#RTL uses exceptions to provide error handling. In event of an error in the C#RTL layer, CJMSException with a specific Error Code and description is thrown.

The Exception can be caught at the application level and the associated message can be read using the public api getMessage(). The exception handling is also exhaustive in that it provides the complete function stack trace at the moment of the error.

The exception stack is maintained in the Thread Local Storage, so that the exception that occurred in one thread doesn't interfere the flow of other threads.

The stack trace can be printed on the console using the API call printStackTrace().

```
try
{
//Application code
}
catch(CJMSException *e)
{
cout << e->getMessage();
e->printStackTrace();
}
```

For more information, read the CJMSException section.