Arcot RiskFort™
Java Developer's Guide
Version 2.2.6

455 West Maude Avenue, Sunnyvale, CA 94085

Arcot RiskFort Developer's Guide
Version 2.2.6
September 2010
Part Number: RF-0226-0DGJ-10

Trademarks

Patents

Arcot Systems, Inc., 455 West Maude Avenue, Sunnyvale, CA 94085

Third Party Software

All the third-party software used by RiskFort and related components are listed in the Appendix G, "Third-Party Software Licenses" of the *Arcot RiskFort 2.2.6 Installation and Deployment Guide*.

Contents

Preface

This guide provides information about how to use Arcot RiskFort Java classes and methods to enable your online application to programmatically perform risk evaluation and related tasks. This document describes the Java implementation of the RiskFort SDK.

Appendix B, "Sample Code", provides a fully-functional sample code that you can run to test the supported user-related operations, risk evaluation, and post-evaluation functionality of RiskFort.

Intended Audience

This guide is targeted at application programmers implementing JAVA programs that require the use of the APIs and functions provided with Arcot RiskFort.

Readers should be familiar with the following:

- Database architecture and concepts
- Security management concepts
- Authentication and authorization concepts
- Internet protocols, including HTTP and TCP/IP
- Secure Sockets Layer (SSL) communications and the related concepts (public and private key exchange, digital certificates and signatures, and certificate authorities)

Information Included in this Guide

This guide is organized in parts as follows:

- Chapter 1, "Getting Started", introduces you to the RiskFort SDK and its features. It also provides a broad outline of the "points" where your application can integrate with RiskFort and walks you through the prerequisites for the integration.
- Chapter 2, "Understanding RiskFort Workflows", covers all the processes and workflows provided by the on-premise RiskFort solution.
- Chapter 3, "Before You Begin", describes what API-specific JAR files and Properties files to include in `CLASSPATH` and how to initialize the APIs.

- Chapter 4, "Managing Users", describes the process of creating a new user in RiskFort, reading their details from the database, and updating these details by using the appropriate classes, interfaces, and methods in the RiskFort Issuance API.
- Chapter 5, "Collecting Device ID and Machine FingerPrint", describes how to implement the RiskFort Device ID and Machine Fingerprint (MFP) collection code through your online application. RiskFort makes extensive use of cookie collection and device fingerprinting to determine the risk associated with end-user logins.
- Chapter 6, "Performing Risk Evaluation", demonstrates how RiskFort performs risk evaluation and post evaluation for a risk analysis request forwarded by your application by using the appropriate classes, interfaces, and methods in the RiskFort Risk Evaluation API.
- Appendix A, "Additional SDK Configurations", provides the details on how to set up multiple RiskFort Server instances and to configure SSL communication between Java SDKs and RiskFort Server.
- Appendix B, "Sample Code", provides a fully-functional sample code that you can run to test the user-related operations, risk evaluation, and post-evaluation functionality of RiskFort.
- Appendix C, "Java API Reference", lists the Issuance and Risk Evaluation Java API files and how to access them.
- Appendix D, "Exceptions and Error Codes", lists the error codes returned by the RiskFort SDK.
- Appendix E, "Input Data Validations", lists the criteria that is used to check the SDK input parameters.
- Appendix F, "RiskFort Logging", covers information about RiskFort log file, the configuration file that controls it, the severity levels recorded, and the format of the entries in the log file.
- Appendix G, "RiskFort Sample Application", introduces you to RiskFort Sample Application, which is a sample Web application that uses RiskFort APIs and demonstrates the most common functionality of RiskFort.
- Appendix H, "Glossary", lists and explains the RiskFort key terms.

Related Publications

Other related publications include:

Arcot RiskFort 2.2.6 Administration Guide	This guide includes the information to administer and configure RiskFort.

Arcot RiskFort 2.2.6 Installation and Deployment Guide	This guide provides information on how to install and configure Arcot RiskFort in single-system and multi-system environments. It is your primary source of information for pre-installation, installation, and post-installation tasks.
Arcot RiskFort 2.2.6 Java Developer's Guide	This guide describes the Java APIs provided by RiskFort and also explains how to use them.
Arcot RiskFort 2.2.6 Quick Installation Guide	This guide provides a summary of the tasks required to install RiskFort.

Conventions Used in This Book

The conventions, formats, and scope of this manual are described in the following paragraphs:

Typographical Conventions

This manual uses the following typographical conventions:

Italic	Emphasis, Guide names
Bold	User input, GUI screen text
`Fixed`	File and directory names, extensions, Command Prompt, CLI text, code in running text
`Fixed Bold`	Target file or directory name in the path
`Fixed`	Command Prompt, CLI text, code
`Fixed-Italic`	File or directory name that might be different from user to user
Link	Links within the guide, URL links

Formats

This manual uses the following formats to highlight special messages:

Note: Highlights information of importance or special interest.

	**Tip:** Highlights a procedure that will save time or resources.
	**Warning:** Ignoring this type of note may result in a malfunction or damage to the equipment.
	**Important:** Information to know before performing an operation.
	**Caution:** Makes the user attentive of the possible danger.
	**Book:** Provides reference to other guides.

Contacting Support

If you need help, contact Arcot Support as follows:

Email	*support@arcot.com*
Web site	*http://www.arcot.com/support/index.html*

Chapter 1
Getting Started

Arcot RiskFort (referred to as RiskFort later in the manual) is an adaptive authentication solution that evaluates each online transaction in real time by examining a wide range of collected data against out-of-box rules. It then assigns each transaction a risk score and advice. The higher the risk score, the greater is the possibility of a fraud. Based on your business policies, your application can then use this risk score and advice to approve or decline the transaction, ask for additional authentication, or alert a customer service representative.

RiskFort offers you the flexibility to modify the configuration parameters of any of the risk evaluation rules in keeping with your policies and risk-mitigation requirements. It also gives you the flexibility to modify the default scoring configuration, scoring priorities, and risk score for any rule, and selectively enable or disable the execution of one or more rules.

Besides pre-configured out-of-the-box rules, RiskFort's field-programmable add-on rules capability allows for industry-specific rules to be selectively deployed and augmented based on your requirements.

Book: See Chapter 1, "Understanding RiskFort Basics" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* to understand the basic concepts of RiskFort and its architecture.

This chapter covers the following topics:

Introduction to the RiskFort SDK

The RiskFort software development kit (SDK) provides a programmatic front-end to RiskFort. It provides APIs for interacting with the RiskFort Server to perform the tasks needed for assessing each login transaction (Risk Evaluation APIs) and for creating users in RiskFort database (Issuance APIs).

After you install RiskFort, the Java SDKs are available at the following location:

Windows:

`<install_location>\Arcot Systems\sdk\java\lib\arcot\`

Unix Platforms:

`<install_location>/arcot/sdk/java/lib/arcot/`

If you have not installed RiskFort yet, you can still access these Javadocs in the `Documentation` directory of the RiskFort package you receive.

Risk Evaluation API

The Risk Evaluation API (`arcot-riskfort-evaluaterisk.jar`) is the interface to RiskFort Server, which provides the logic for evaluating risk associated with a transaction and returning an appropriate advice.

Based on the various factors collected from user's system and the result of configured rules that are triggered, this API returns a score and a corresponding advice. In case RiskFort recommends secondary authentication (which is performed by your application), it also returns a final advice based on the feedback of this secondary authentication received from your application.

In addition, by using a cookie or a Flash Shared Object (FSO), this API associates (or binds) a user to the device that they use to access your application. This *association* (or device binding) helps identify the risk for transactions originating from a system used by the user for a transaction. Users who are not bound are more likely to be challenged in order to be authenticated. You can also list and delete these associations by using this API.

Note: Users can be bound to more than one device (for example, someone using a work and home computer) and a single device can be bound to more than one user (for example, a family using one computer).

Issuance API

The RiskFort Issuance API (`arcot-riskfort-issuance.jar`) enables you to create, list, and update user details in the RiskFort database. By creating users in the RiskFort database, you enable RiskFort to track a user's risk history and use it for further risk analysis.

RiskFort SDK Features

This section discusses the salient features of the Risk Evaluation and Issuance SDKs.

- **Multiple Ways to Initialize the SDKs**

  You can initialize Risk Evaluation and Issuance SDKs either by using properties file or by using a map. See Chapter 3, "Before You Begin" for more information on how to do this.

- **Support for SDK Failover**

  Java SDKs support failover mechanism, if an instance of RiskFort Server is not operational, then the SDKs automatically connect to any of the additional configured instances. See "Configuring Multiple RiskFort Server Instances" on page A-73 for more information on how to do this.

- **Support for SSL**

  You can secure the connection between the Java SDK and RiskFort Server by using Secure Socket Layer (SSL). To set up SSL between SDK and WebFort Server, you must edit the RiskFort properties files. See "Setting Up SSL Communication Between SDKs and RiskFort Server" on page A-74 for more information on how to do this.

- **Support for Additional Parameters**

  In addition to the mandatory inputs, the APIs also accepts additional input that can be passed as `name-value` pair. This input can include information, such as locale, calling application details, or additional transaction details.

Overview of the Integration Steps

The RiskFort SDK offers you multiple degrees of freedom in the available integration methods and the types of risk-based authentication flows. (See Chapter 2, "Understanding RiskFort Workflows" for more information on supported workflows.) This enables you to design the optimal authentication solution that best suits your organization's requirements.

The RiskFort flows can be integrated with your online application at the points discussed in following subsections.

Before a User Logs in to Your Application (and Just Accesses the Login Page)

In this case, your application can invoke RiskFort's `evaluateRisk()` function call from the login page (*before* the user specifies the login credentials) to assess the risk associated with the incoming data. For example, you can evaluate the IP address and/or the country for Negative IP and Negative Country checks.

Note: Negative IP addresses is a collection of IP addresses that have been the origin of known anonymizer proxies or fraudulent or malicious transactions in past. Similarly, Negative countries is a collection of all countries from which fraudulent or malicious transactions have been recorded in past.

In this case, you can also evaluate other RiskFort rules that do not require user information. These include, Device Velocity Check, Zone Hopping Check, and any custom Add-On rules you might have implemented.

After a User Logs in to Your Online Application (By Specifying the User Name and Password to Access Their Account or the Protected Resource)

In this case:

1. Your application must invoke RiskFort from the main page of your application that appears after successful login. The following scenarios are possible:
   - **User is enrolled with RiskFort**, and must undergo risk evaluation.

     In this case your application must invoke RiskFort's `evaluateRisk()` function call by passing user, device signature (collected by using the MFP Javascript provided by RiskFort), IP address, and transaction details for assessing the risk:
     - If the received user data is assigned a low score after rule execution (based on the incoming data and the data stored for this user or device), then the advice is `ALLOW`.

In this case, your application must grant access to the user to the protected resource or Web page and allow the user to continue with the transaction.

- If the received user data is assigned a high score after rule execution (based on the incoming data and the data stored for this user or device), then the advice is `DENY`.

  In this case, your organizational policies determine the outcome. The transaction can be denied or can be forwarded to a security analyst (known as Customer Support Representative (CSR) in RiskFort terminology) for review and further action.

- If a transaction is flagged as suspicious, then the advice is `INCREASEAUTH`.

  In this case, your application must perform a secondary authentication, which can include industry-standard security questions (such as mother's maiden name and date of birth).

Note: You can also use Arcot WebFort for this purpose.

See *http://www.arcot.com/* for more information.

Only if your application authenticates the user during the secondary authentication, then you must grant access to the user to the protected resource or Web page and allow the user to continue with the transaction.

- **User is new to RiskFort**, and therefore must be enrolled with it.

  In this case, your application must invoke RiskFort's `createUser()` function call by passing user details to create the user in RiskFort database.

Important: Because RiskFort works "behind the scene" for an end user, Arcot strongly recommends that you do not change the end-user experience for this enrollment.

After RiskFort "knows" the user after this enrollment, then you must call the `evaluateRisk()` function and allow the user to proceed with the transaction or access the protected resource, if the advice is `ALLOW`.

2. Your application now must invoke RiskFort's `postEvaluate()` function call. By using this call, RiskFort generates the final advice and creates an association, if the result of `evaluateRisk()` was `INCREASEAUTH`. In case of other advices, it just updates the associations and attributes. This data serves as the feedback to be used for risk analysis of future transactions.

Before You Begin

Before you integrate your application with RiskFort, RiskFort must be installed and configured, ensuring that:

- The systems on which you plan to install RiskFort meets the system requirements.

Book: Refer to the section, "System Requirements" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide*.

- You have completed the configuration and planning-related information:
  - You have installed and configured the required number of RiskFort database instances.

Book: See sections, "Configuring Database Server" and "Database-Related Post-Installation Tasks" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for detailed instructions.

  - You have installed the applicable version of JDK on the system where you plan to install RiskFort components that use JDK.
  - You have also installed the required Application server.

Book: See the section, "Requirements for Java-Dependent Components" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide*.

In case of **single-system deployment** of RiskFort (see Chapter 4, "Deploying RiskFort on a Single System" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for more information), ensure that the all the components are up.

In case of **distributed-system deployment** of RiskFort (see Chapter 5, "Deploying RiskFort on a Distributed System" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for more information), ensure that the connection is established between all the components and that they communicate with each other.

Chapter 2
Understanding RiskFort Workflows

RiskFort provides many workflows that can be integrated and used by your online application. Based on your organizational requirements, you can integrate these workflows, without changing the existing online experience for your users in most cases, except when RiskFort generates the `INCREASEAUTH` advice.

This chapter describes these workflows and provides an overview of each workflow so you can understand the different processes involved:

Enrollment Workflows

Every time your application forwards a request for risk analysis, RiskFort uses the **Unknown User Check** rule to determine if the user details exist in the RiskFort database. If this information is not found, then RiskFort treats the incoming request as a first-time (or unknown) user request and recommends the `ALERT` advice. In such cases, you must enroll the user.

Enrollment is the process of creating a new user in the RiskFort database. As discussed in the following subsections, you can enroll the user explicitly by calling the `createUser()` method of the Issuance SDK from your application. You can also create the user implicitly by selecting the **Implicit** option in the Miscellaneous Rules Configuration screen of the Administration Console.

After enrollment, you must perform risk evaluation (as discussed in "Risk Evaluation Workflows" on page 2-15), which creates the user implicitly in the system. However if the user is not registered with your application, then you must take action according to your organizational policies.

Explicit Enrollment

In case of *explicit enrollment*, you must call RiskFort's `createUser()` function explicitly from your application's code to create a user in RiskFort database. You can call this function either *before* (Scenario 1) or *after* (Scenario 2) you perform risk evaluation (by using the `evaluateRisk()` call.)

Scenario 1

The steps for the explicit enrollment workflow, if you call the `createUser()` function before `evaluateRisk()` function are:

1. **User logs into your online application.**

   Your system validates if the user exists in your system. If the user name is not valid, then your application must take appropriate action.

2. **Your application calls RiskFort's `createUser()` function.**

   At this stage, your application must make an explicit call to the `createUser()` function in `riskfortissuanceAPI`. In this call, you must pass all pertinent user details, such as user's name, last name, organization, email, and their personal assurance message (PAM) to RiskFort.

3. **RiskFort creates the user in the database.**

   If the `createUser()` call was successful, then RiskFort creates the user record in the RiskFort database. With this, user is enrolled with RiskFort.

4. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.
   - (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

5. **You application calls RiskFort's `evaluateRisk()` for risk analysis.**

   In this case, because you enrolled the user *before* performing risk analysis, the RiskFort system "knows" the user and does not generate the `ALERT` advice. Refer to "Risk Evaluation Workflows" for more information.

6. **RiskFort performs risk analysis.**

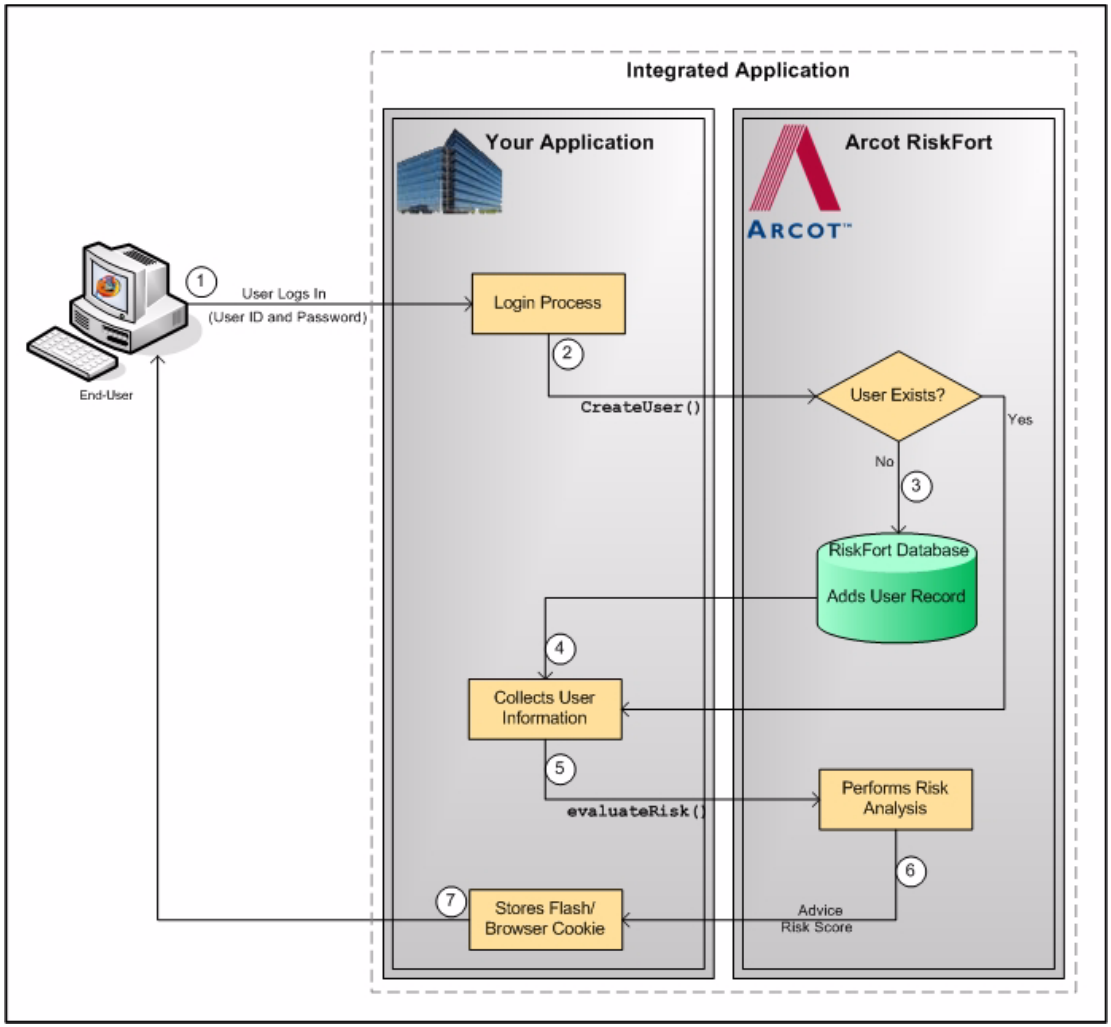
   RiskFort generates a risk score and an advice.

7. **Your application stores the DeviceID on the end user's system.**

Your application must store the Device ID returned by `evaluateRisk()` as a cookie (FSO or browser cookie) on the device that user is using for the current transaction.

Figure 2-1 illustrates the explicit enrollment workflow when you call `createUser()` before the `evaluateRisk()` call.

Figure 2-1 Explicit Enrollment Workflow: Calling createUser() *Before* evaluateRisk()



Scenario 2

The steps for the explicit enrollment workflow, if you call the `createUser()` function after `evaluateRisk()` function, are:

1. **User logs into your online application.**

   Your system validates if the user exists in your system. If the user name is not valid, then your application must take appropriate action.

2. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.
   - (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

3. **Your application calls RiskFort's `evaluateRisk()` function.**

   At this stage, your application must call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must pass all the user and device information that you collected in the preceding step to RiskFort.

4. **You application calls RiskFort's `evaluateRisk()` for risk analysis.**

   RiskFort performs risk analysis for the user and generates an advice. In this case because the user is not yet "known" to the RiskFort system, the `ALERT` advice is generated.

5. **Your application calls RiskFort's `createUser()` function.**

   For an `ALERT` advice that is generated, your application must make an explicit call to the `createUser()` function in `riskfortissuanceAPI`. In this call, you must pass all pertinent user details, such as user's name, last name, organization, email, and their personal assurance message (PAM) to RiskFort.

6. **RiskFort creates the user in the database.**

   If the `createUser()` call was successful, then RiskFort creates the user record in the RiskFort database. With this, user is enrolled with RiskFort.

7. **Your application calls RiskFort's `evaluateRisk()` function again.**

   At this stage, your application must again call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must ensure that you pass all the user and device information that you collected in Step 2 to RiskFort.

8. **RiskFort performs risk analysis for the user.**

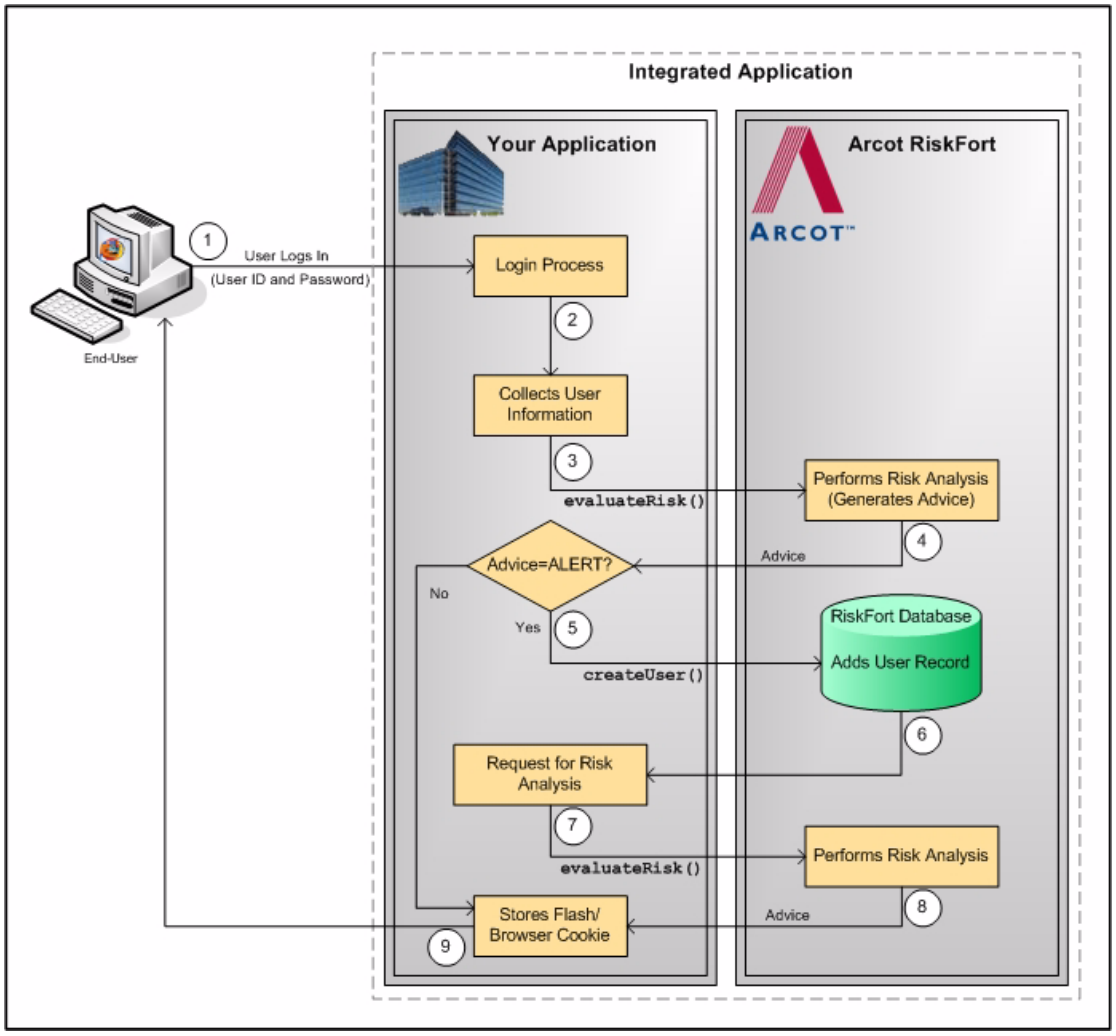
   In this case, RiskFort executes the rules and generates the risk score and the advice.

9. **Your application stores the DeviceID on the end user's system.**

   Your application must store the Device ID returned by `evaluateRisk()` as a cookie (FSO or browser cookie) on the device that user is using for the current transaction.

Figure 2-2 illustrates the explicit enrollment workflow when you call `createUser()` before the `evaluateRisk()` call.

Figure 2-2 Explicit Enrollment Workflow: Calling createUser() *Before* evaluateRisk()



Implicit Enrollment

In case of *implicit enrollment*, you do not need to call RiskFort's `createUser()` function explicitly from your application's code to create a user in RiskFort database. Instead when RiskFort generates the `ALERT` advice for an "unknown user", it automatically calls the function to enroll the user.

For this enrollment to work, it is important that you first set the **User Creation Mode** option on the Miscellaneous Rules Configuration page of the Administration Console to **Implicit**.

The steps for the implicit enrollment workflow are:

1. **User logs into your online application.**

   Your system validates if the user exists in your system. If the user name is not valid, then your application must take appropriate action.

2. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.
   - (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

3. **Your application calls RiskFort's `evaluateRisk()` function.**

   At this stage, your application must call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must pass all the user and device information that you collected in the preceding step to RiskFort.

4. **RiskFort performs risk analysis for the user.**

   In this case because the user is not yet "known" to the RiskFort system, the default `ALERT` advice is generated.

5. **RiskFort creates the user in database.**

   For an `ALERT` advice that is generated, RiskFort uses the `createUser()` function to create the user record in the RiskFort database. With this, user is enrolled with RiskFort.

6. **Your application calls RiskFort's `evaluateRisk()` function again.**

   At this stage, your application must again call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must ensure that you pass all the user and device information that you collected in Step 2 to RiskFort.

7. **RiskFort performs risk analysis for the user.**

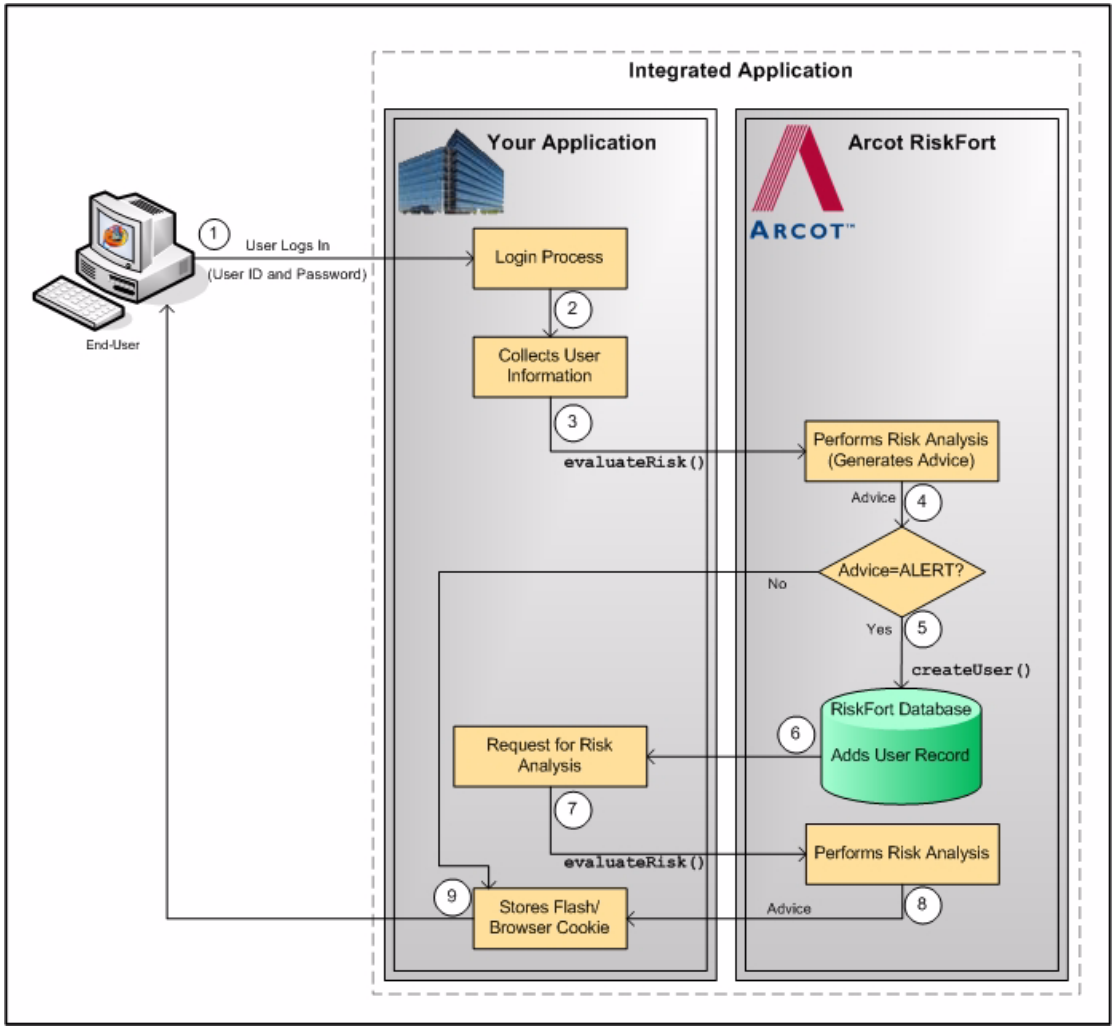
In this case, RiskFort executes the rules and generates the risk score and the advice.

8. **Your application stores the DeviceID on the end user's system.**

   After the user has been created, your application must store the DeviceID (FSO or browser cookie) returned by `evaluateRisk()` as a cookie on the device that user is using for the current transaction.

Figure 2-3 illustrates the implicit enrollment workflow when RiskFort automatically creates the user.

Figure 2-3 Implicit Enrollment Workflow



Risk Evaluation Workflows

The risk evaluation workflows enable your online application to determine if the incoming user request is potentially risky:

- If the risk is low, the user is allowed to access your online application.
- If the risk is high, the user is denied access to your system.
- If the transaction is tagged as suspicious, this workflow also prompts your application to challenge users for additional (secondary) authentication to prove their identity.

You can implement RiskFort's risk analysis capability either before the user logs in your online application or after they have successfully logged in. Depending on when you call RiskFort's `evaluateRisk()` function, the following workflows are possible:

- Pre-Login Risk Evaluation Workflow
- Post-Login Risk Evaluation Workflow

Pre-Login Risk Evaluation Workflow

When a user accesses your online application, you can assess them for potential risk even before they log in by implementing this workflow. This workflow will only use inputs related to device identification and location information (such as IP address, Device ID, and MFP) and rules that do not require user-specific information as the criterion for risk evaluation.

If you call RiskFort's risk analysis capability even before a user logs in to your online application, then the risk evaluation workflow is as follows:

1. **User accesses your online application.**

   When a user accesses your online application, you can assess them for potential risk even before they log in.

2. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.

- (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

3. **Your application calls RiskFort's `evaluateRisk()` function.**

   At this stage, your application must call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must pass the information that you collected in the preceding step to RiskFort.

4. **RiskFort performs risk analysis for the user.**

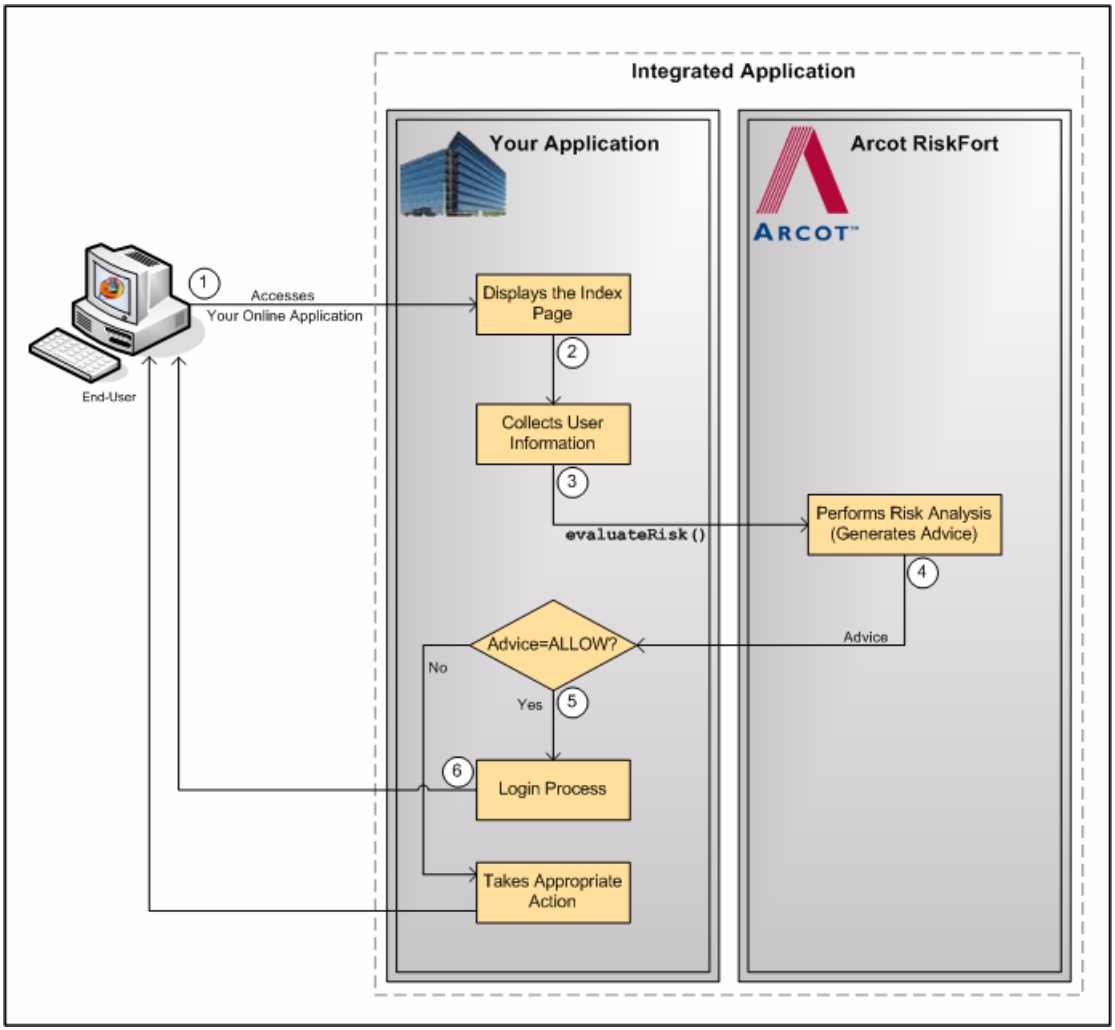
   RiskFort generates the appropriate risk score and advice based on the passed user inputs and configured rules.

5. **Your application validates the user.**

   Based on RiskFort's recommendation, your application can allow the user to proceed with the login process or can deny access to your system.

Figure 2-4 illustrates the Pre-login risk evaluation workflow.

Figure 2-4 Pre-Login Risk Evaluation Workflow



Post-Login Risk Evaluation Workflow

When a user accesses your online application, you can first log them in and then comprehensively assess them for potential risks by implementing this workflow. This workflow uses device identification information and number of factors, such as network information, user information, and (if implemented) transaction information to evaluate users.

Based on the risk score and the subsequent advice, device information is then updated in the RiskFort database:

- In case of `ALLOW`, the user-device association information is updated.
- In case of `ALERT` and `DENY`, the user-device association information is *not* updated at all.
- In case of `INCREASEAUTH`, the user-device association information is updated, but the user association information is created *only if* the result of the additional authentication ("Secondary Authentication Workflow" on page 2-20) was successful.

If you call RiskFort's risk analysis capability after you authenticate a user in to your online application, then the risk evaluation workflow is as follows:

1. **User logs into your online application.**

   Your system validates if the user exists in your system. If the user is not valid, then your application must take appropriate action.

2. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.
   - (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

3. **Your application calls RiskFort's `evaluateRisk()` function.**

   At this stage, your application must call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must pass all the user and device information that you collected in the preceding step to RiskFort.

4. **RiskFort performs risk analysis for the user.**

   RiskFort evaluates the risk using the incoming inputs and the configured rules. Based on the result of rules that were executed and whether the information matched, RiskFort generates:

   - `ALERT`, if the information for the user does not exist in the RiskFort database.
   - `ALLOW`, if the information match confidence is high.

- `DENY`, if the information match confidence is low.
- `INCREASEAUTH`, if the incoming information is suspicious.

If the advice is `INCREASEAUTH`, then refer to "Secondary Authentication Workflow" for more information on how to proceed.

5. **Your application takes the appropriate action using RiskFort's recommendation.**

   Based on the result of the `evaluateRisk()` call, your application either allows the user to continue with the transaction, denies them access to the protected resource, or performs secondary authentication.

   See "Secondary Authentication Workflow" on page 2-20 for more information.

6. **Your application calls RiskFort's `postEvaluate()` function.**

   At this stage, your application must call the `postEvaluate()` function in `riskfortAPI`. Based on the output generated by the `evaluateRisk()` call, this call helps RiskFort to generate the final advice and update the device and association information.

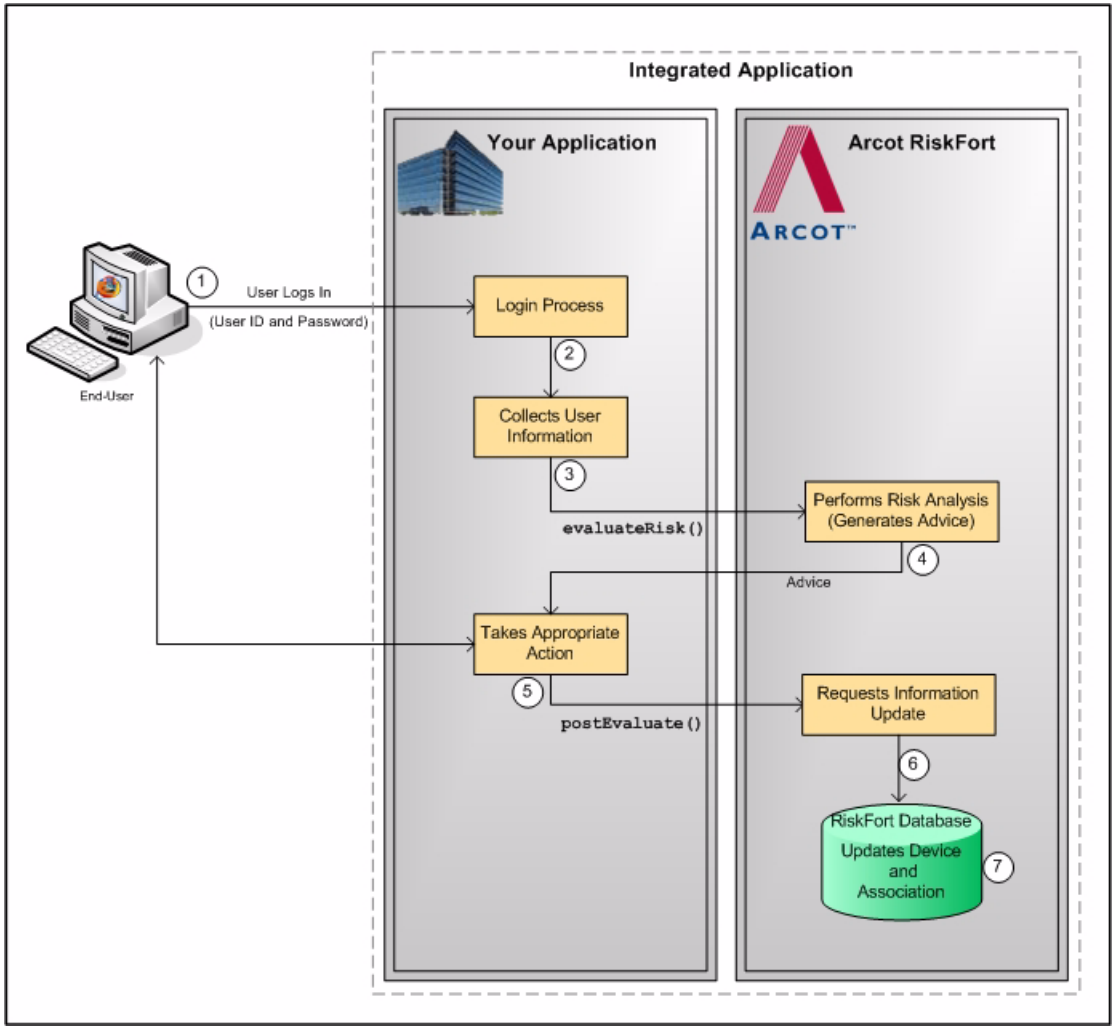
   In this call, you must pass the risk score and advice from the `evaluateRisk()` call, the result of secondary authentication (if the advice in the previous step was `INCREASEAUTH`), and any association name, if the user specified one.

7. **RiskFort updates the device and association information.**

   If any change is detected in the incoming data, RiskFort updates the data and association information in the RiskFort database.

Figure 2-5 illustrates the Post-login risk evaluation workflow.

Figure 2-5 Post-Login Risk Evaluation Workflow



Secondary Authentication Workflow

When RiskFort generates the `INCREASEAUTH` advice, it transfers the control back to your application temporarily for secondary authentication. In this case, your application must implement some mechanism for performing additional authentication. For example, your application can display industry-standard security (or challenge) questions to the user (such as mother's maiden name and date of birth) or make them undergo out-of-band phone authentication.

After you determine whether the user authenticated successfully or not, you must forward the result to RiskFort, which uses this feedback to generate the final advice, update device information, create association information, and to store the feedback to use for risk analysis of future transactions.

The risk evaluation workflow in case of secondary authentication is as follows:

1. **User logs into your online application.**

   Your system validates if the user exists in your system. If the user is not valid, then your application must take appropriate action.

2. **Your application collects information required by RiskFort.**

   At this stage, your application must use RiskFort's Utility Script called `json.js` to collect information from the user's system that will be used by RiskFort for analyzing risk:

   - **User system information** that includes operating system, platform, browser information (such as browser language, HTTP header information), locale, and screen settings.
   - **Device information** that includes DeviceID, such as Flash Shared Object (FSO) cookie or browser cookie.
   - **Location information** that includes IP address and ISP-related information.
   - (***Optionally, if you are using additional information***) **Additional Inputs** that might include transaction amount, locale, and related information.

3. **Your application calls RiskFort's `evaluateRisk()` function.**

   At this stage, your application must call the `evaluateRisk()` function in `riskfortAPI`. In this call, you must pass all the user and device information that you collected in the preceding step to RiskFort.

4. **RiskFort performs risk analysis for the user.**

   If RiskFort flags the transaction as suspicious, it generates the `INCREASEAUTH` advice. This implies that extra credentials are required to help further authenticate the user.

5. **Your application performs secondary authentication.**

   Based on the secondary authentication mechanism that you are using, your application displays appropriate pages to the user. For example, you can prompt the user to:

   - Answer the security questions that they selected while enrolling with your application.
   - Perform One-Time Password (OTP) authentication.
   - Perform phone authentication.

After receiving user input, your application determines the outcome of the additional authentication.

6. **Your application calls RiskFort's `postEvaluate()` function and forwards the result of the secondary authentication to RiskFort.**

   At this stage, irrespective of the fact whether the user failed or cleared the secondary authentication, your application *must* pass the result back to RiskFort. This information helps RiskFort build an up-to-date and accurate user history.

   To do so, your application must call the `postEvaluate()` function in `riskfortAPI`. In this call, you must pass the risk score and advice from the `evaluateRisk()` call, the result of secondary authentication, and any association name, if the user specified one.

7. **RiskFort generates the final advice.**

   By using your application's feedback regarding the secondary authentication, RiskFort generates the final advice.

8. **RiskFort updates the device information and creates the association information.**

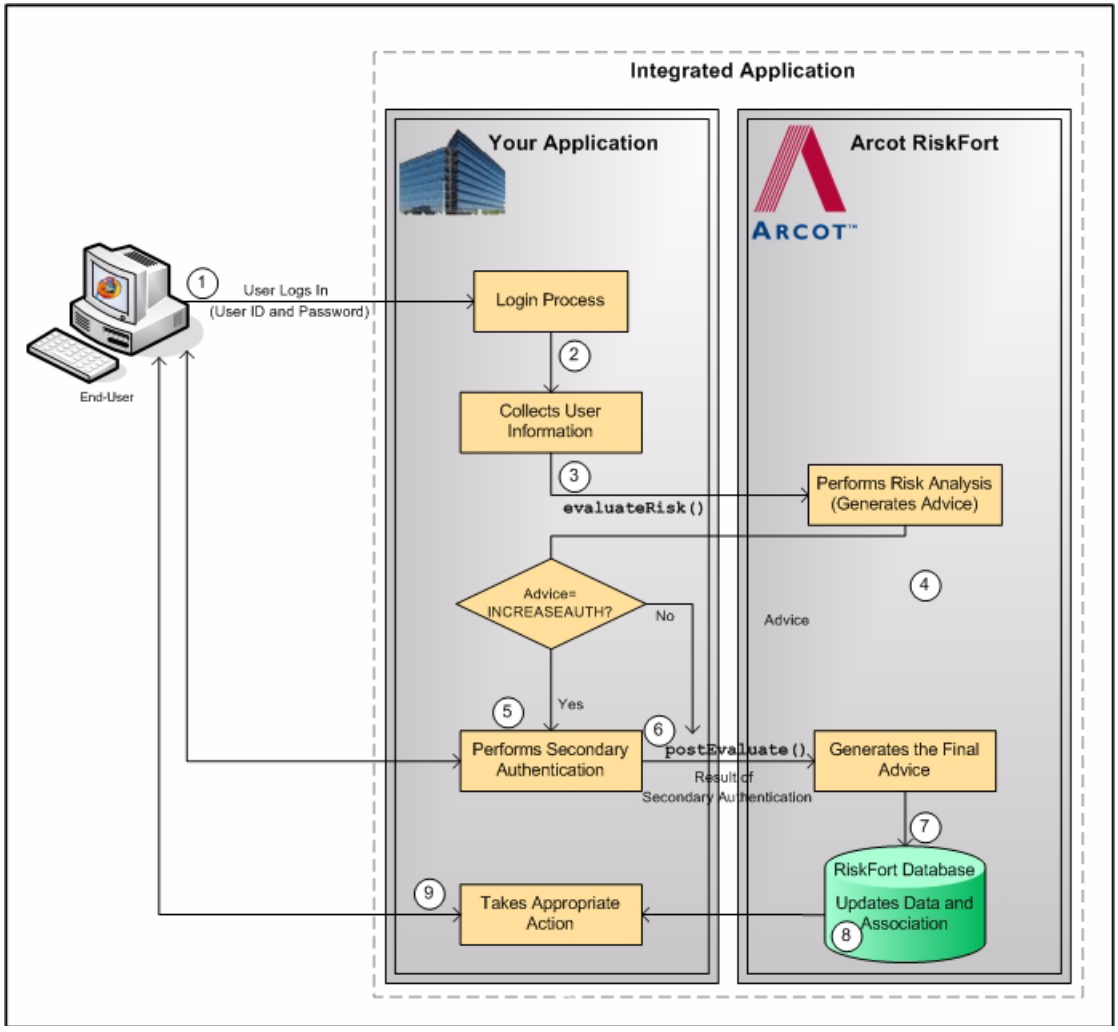
   Based on the result of the `postEvaluate()` call, RiskFort also updates the device attributes and creates the association information in the RiskFort database.

9. **Your application takes the appropriate action.**

   Based on the result of the `postEvaluate()` call, your application either allows the user to continue with the transaction or denies them access to the protected resource.

Figure 2-6 illustrates the secondary authentication risk evaluation workflow.

Figure 2-6 Secondary Authentication Risk Evaluation Workflow



Workflow Summary

Table 2-1 provides a brief summary of the workflows provided by RiskFort.

Note: All these workflows, except for secondary authentication workflow, are implemented "behind the scenes" and do not change the user experience.

Table 2-1. Summary of RiskFort Workflows

Workflow	Sub-Type of the Workflow	Description	Dependant Workflows
Enrollment	**Explicit**	**Scenario 1:** Creates a user in the RiskFort database, when you call `createUser()` **before** `evaluateRisk()`. In this case, the end user never gets an `ALERT` advice.	• Post-Login Risk Evaluation
		Scenario 2: Creates a user in the RiskFort database, when you call `createUser()` **after** `evaluateRisk()`.	• Post-Login Risk Evaluation
	Implicit	RiskFort **automatically** creates a user in the RiskFort database, without you having to make the `createUser()` call.	• Post-Login Risk Evaluation
Risk Evaluation	**Pre-Login**	Analyzes the risk of a transaction before the user logs in to your online application system.	None
	Post-Login	Analyzes the risk of a transaction after the user logs in to your online application system. Also updates user information and device association information.	• Enrollment • Secondary Authentication
	In case of **Secondary Authentication**	Provides the final advice in case your application performed a secondary authentication after RiskFort recommended `INCREASEAUTH`. Also updates user information and device association information.	• Post-Login Risk Evaluation

Chapter 3
Before You Begin

Before you use the Issuance or Risk Evaluation API, you must include the related JAR files in the `CLASSPATH`. If you are using Properties files in your application, then you must also include them in the `CLASSPATH`. After including the required JAR files or Properties files in `CLASSPATH`, you must next Initialize the API.

The chapter covers following topics:

- Before You Integrate with the Issuance API
- Before You Integrate with Risk Evaluation API

Before You Integrate with the Issuance API

This section covers the following topics:

- Including Issuance JAR Files in CLASSPATH
- Including Properties Files in CLASSPATH
- Initializing the Issuance API
- Preparing Additional Inputs

Including Issuance JAR Files in CLASSPATH

To use the API, you need to include the appropriate Issuance JAR files in the `CLASSPATH` variable. To do so:

1. If required, you need to copy the Issuance JAR files listed in the following tables to the appropriate directory (say `lib`) in your <*APPLICATION_CONTEXT*>.

For Windows

Table 3-1. Issuance JAR Files (Windows) to be Included in CLASSPATH

Primary JAR Files	arcot_core.jar arcot-riskfort-issuance.jar	<*install_location*>\Arcot Systems**sdk****java****lib****arcot**\
Other JAR Files	bcprov-jdk14-139.jar commons-beanutils-1.7.0.jar commons-collections-3.1.jar commons-httpclient-3.1.jar commons-lang-2.0.jar commons-logging-1.0.4.jar commons-pool-1.4.jar dom4j-1.6.1.jar jaxen-1.1-beta-8.jar jdom-1.0.jar json-lib-0.7.1.jar log4j-1.2.9.jar servlet-api-2.4.jar oro-2.0.8.jar xalan-2.7.0.jar xercesImpl-2.6.2.jar xml-apis-1.0.b2.jar xmlParserAPIs-2.6.2.jar xom-1.1.jar	<install_location>\Arcot Systems**sdk****java****lib****external**\

For UNIX-Based Platforms

Table 3-2. Issuance JAR Files (UNIX Platforms) to be Included in CLASSPATH

Primary JAR Files	arcot_core.jar arcot-riskfort-issuance.jar	*<install_location>*/arcot/**sdk/java/lib/arcot/**
Other JAR Files	bcprov-jdk14-131.jar commons-beanutils.jar commons-collections-3.1.jar commons-digester.jar commons-lang-2.0.jar commons-logging.jar commons-pool.jar log4j-1.2.9.jar servlet.jar sqljdbc.jar	*<install_location>*/arcot/**sdk/java/lib/external/**

2. Add the parent directory (in the *<APPLICATION_CONTEXT>*) of the directory to which you copied the JARs to the CLASSPATH environment variable.

Including Properties Files in CLASSPATH

If your application is using Properties files, then the riskfort.issuance.properties file must be included in CLASSPATH. To ensure this:

1. Copy the properties directory from the following location to the appropriate directory (say classes) in your *<APPLICATION_CONTEXT>*.

   - **For Windows**

   <install_location>\Arcot Systems**sdk****java**\

   - **For UNIX Platforms**

   <install_location>/arcot/**sdk/java/**

2. Add the parent directory of the properties directory to the CLASSPATH environment variable.

Initializing the Issuance API

Initialize the Issuance API by using the `UserRepManager` class in the `com.arcot.riskfortissuanceAPI` package. The initialization process creates all connection pools, creates the database pool, and initializes loggers.

Note: You *cannot* apply any configuration changes after you initialize the API. To enable the configuration changes, you must re-initialize the API.

The `UserRepManager` class provides two methods, as discussed in the following subsections, to initialize the Issuance APIs.

Method 1:

Initializing the API by Using the Properties File

The `initialize(java.lang.String propertyLocation)` method initializes the Issuance API by using the parameters listed in the input Properties file. If you pass `NULL`, then the parameters are read from the `riskfort.issuance.properties` file, which is present in the `properties` folder of the `CLASSPATH`.

The fields and format of the Properties file must be as follows:

```
//RiskFort Server IP address
HOST.1=

//RiskFort Server port number
PORT.1=

//Type of the connection. Possible values include TLS and TCP.
TRANSPORT_TYPE=

//Required if TRANSPORT_TYPE is set to TLS. The file must be in
//PEM format. Provide the complete path for the file.
CA_CERT_FILE=
```

Table 3-3 provides the details of the `initialize()` method.

Table 3-3. API Initialization by Using Issuance Properties File

Description	Input Parameter	Output Value
Initializes the Issuance API by using the specified Properties file.	• `propertyLocation` The absolute path of the Properties file.	Throws `UserRepositoryException` if the API was not initialized successfully.

Method 2:

Initializing the API by Using the Map

The `initialize(java.util.Map properties)` method in the `UserRepManager` class initializes the Issuance API based on the map provided.

Table 3-4 provides the details of the `initialize()` method.

Table 3-4. API Initialization by Using Map

Description	Input Value	Output Value
Initializes the Issuance API by using the provided map.	• `map` The key-value pair specifying the configuration information. The supported keys are: •`HOST.1` The IP address of the host where RiskFort Server is available. •`PORT.1` The port at which RiskFort Server is available. Default value is `7680`. •`TRANSPORT_TYPE` The type of the connection. Possible values include SSL and TCP. •`CA_CERT_FILE` (Required only if `TRANSPORT_TYPE` is set to `SSL`.) The path for the CA certificate file of the server. The file must be in PEM format. Provide the complete path for the file.	Throws `UserRepositoryException` if the API was not initialized successfully.

Preparing Additional Inputs

In addition to the mandatory inputs, the Issuance API also accepts additional inputs in form of `name-value` pairs. This input can include information, such as locale, calling application details, or other user-related details. These additional `name-value` custom parameters can help you capture the real-time inputs from each transaction, and are processed by deployed Add-On rules for successful user creation in RiskFort database.

RiskFort's `com.arcot.riskfortissuanceAPI.AdditionalInputs` package provides you the `AdditionalInputs` class, which enables you to set the additional information that you plan to use. Some of the pre-defined additional input parameters supported by the Issuance `AdditionalInputs` class include:

- `AR_RF_LOCALE_ID`

  Specifies the locale that RiskFort will use while returning the messages back to your calling application.

- `AR_RF_CALLER_ID`

  Specifies the transaction identifier in your calling application. This is useful for end-to-end tracking of transactions.

Tip: *Any* parameter can be added to the `AdditionalInputs` class and can then be passed to the Add-On rules or Evaluation Callout.

To implement custom Issuance parameters:

1. Use the `AdditionalInputs()` method to obtain an object that implements the `AdditionalInputs` class.
2. Set the necessary inputs for the returned object by using the `put()` method. The syntax of the method is:

   ```
   public void put(java.lang.String name, java.lang.String value)
   ```

   If the additional input is *not* present with the given `name`, then one is created. Otherwise, it is overwritten with the new `value`. It is recommended that large strings are not used either for `name` or for `value`.

Note: The `name` and `value` parameters must *not* contain = and the newline character (`\n`). The API behavior is undefined if `name` or `value` contain any of these characters.

Before You Integrate with Risk Evaluation API

This section covers the following topics:

Including Risk Evaluation JAR Files in CLASSPATH

To use the API, you need to include the appropriate Risk Evaluation JAR files in the `CLASSPATH` variable. To do so:

1. If required, you need to copy the Risk Evaluation JAR files listed in the following tables to the appropriate directory (say `lib`) in your <*APPLICATION_CONTEXT*>.

For Windows

Table 3-5. Risk Evaluation JAR Files (Windows) to be Included in CLASSPATH

Primary JAR Files	arcot_core.jar arcot-riskfort-evaluaterisk.jar arcot-riskfort-mfp.jar	<*install_location*>\Arcot Systems**sdk****java****lib****arcot**\
Other JAR Files	bcprov-jdk14-131.jar commons-beanutils.jar commons-collections-3.1.jar commons-digester.jar commons-lang-2.0.jar commons-logging.jar commons-pool.jar log4j-1.2.9.jar servlet.jar sqljdbc.jar	<*install_location*>\Arcot Systems**sdk****java****lib****external**\

For UNIX-Based Platforms

Table 3-6. Risk Evaluation JAR Files (UNIX) to be Included in CLASSPATH

Primary JAR Files	arcot_core.jar arcot-riskfort-evaluaterisk.jar arcot-riskfort-mfp.jar	<*install_location*>/arcot/**sdk**/**java**/**lib**/**arcot**/
Other JAR Files	bcprov-jdk14-131.jar commons-beanutils.jar commons-collections-3.1.jar commons-digester.jar commons-lang-2.0.jar commons-logging.jar commons-pool.jar log4j-1.2.9.jar servlet.jar sqljdbc.jar	<*install_location*>/arcot/**sdk**/**java**/**lib**/**external**/

2. Add the parent directory (in the <*APPLICATION_CONTEXT*>) of the directory to which you copied the JARs to the CLASSPATH environment variable.

Including Properties Files in CLASSPATH

If your application is using Properties files, then the riskfort.risk-evaluation.properties file must be included in CLASSPATH. To ensure this:

1. Copy the properties directory from the following location to the appropriate directory (say classes) in your <*APPLICATION_CONTEXT*>.

   - **For Windows**

     <*install_location*>\Arcot Systems**sdk****java**\

   - **For UNIX Platforms**

     <*install_location*>/arcot/**sdk**/**java**/

2. Add the parent directory of the properties directory to the CLASSPATH environment variable.

Initializing the Risk Evaluation API

Initialize the Risk Evaluation API by using the RiskFactory class in the com.arcot.riskfortAPI package. The initialization process creates all connection pools, creates the database pool, and initializes loggers. After initialization, it returns an appropriate object to your calling application.

Note: You *cannot* apply any configuration changes after you initialize the API. To enable the configuration changes, you must re-initialize the API.

The RiskFactory class provides two methods, as discussed in the following subsections, to initialize the Risk Evaluation APIs.

Method 1:

Initializing the API by Using the Properties File

The `initialize(java.lang.String propertyLocation)` method initializes the Risk Evaluation API by using the parameters listed in the input Properties file. If you pass `NULL`, then the parameters are read from the `riskfort.risk-evaluation.properties` file, which is present in the `properties` folder of the `CLASSPATH`.

The fields and format of this properties file must be as follows:

```
//RiskFort Server IP address
HOST.1=

//RiskFort Server port number
PORT.1=

//Type of the connection. Possible values include TLS and TCP.
TRANSPORT_TYPE=

//Required if TRANSPORT_TYPE is set to TLS. The file must be in
//PEM format. Provide the complete path for the file.
CA_CERT_FILE=
```

Table 3-7 provides the details of the `initialize()` method.

Table 3-7. API Initialization by Using Risk Evaluation Properties File

Description	Input Parameter	Output Value
Initializes the Issuance API by using the specified Properties file.	• `propertyLocation` The absolute path of the Properties file.	Throws `UserRepositoryException` if the API was not initialized successfully.

Method 2:

Initializing the API by Using the Map

The `initialize(java.util.Map properties)` method in the `RiskFactory` class initializes the Risk Evaluation API based on the map provided. Table 3-8 provides the details of the `initialize()` method.

Table 3-8. API Initialization by Using Map

Description	Input Value	Output Value
Initializes the Issuance API by using the provided map.	• `map` The key-value pair specifying the configuration information. The supported keys are: •`HOST.1` The IP address of the host where RiskFort Server is available. •`PORT.1` The port at which RiskFort Server is available. Default value is `7680`. •`TRANSPORT_TYPE` The type of the connection. Possible values include SSL and TCP. •`CA_CERT_FILE` (Required only if `TRANSPORT_TYPE` is set to `SSL`.) The path for the CA certificate file of the server. The file must be in PEM format. Provide the complete path for the file.	Throws `UserRepositoryException` if the API was not initialized successfully.

Preparing Additional Inputs

In addition to the mandatory inputs, the Risk Evaluation API also accepts additional inputs in form of `name-value` pairs. This input can include information, such as locale, calling application details, or other transaction-related details that can be used for risk evaluation. These additional `name-value` custom parameters can help you capture the real-time inputs from each transaction, and are processed by deployed Add-On rules for successful risk evaluation.

RiskFort's `com.arcot.riskfortissuanceAPI.AdditionalInputs` package provides you the `AdditionalInputs` class, which enables you to set the additional information that you plan to use. Some of the pre-defined additional input parameters supported by the Risk Evaluation `AdditionalInputs` class include:

- `AR_RF_LOCALE_ID`

  Specifies the locale that RiskFort will use while returning the messages back to your calling application.

- `AR_RF_CALLER_ID`

  Specifies the transaction identifier in your calling application. This is useful for end-to-end tracking of transactions.

Tip: *Any* parameter can be added to the `AdditionalInputs` class and can then be passed to the Add-On rules or Evaluation Callout.

To implement custom risk evaluation parameters:

1. Use the `AdditionalInputs()` method to obtain an object that implements the `AdditionalInputs` class.
2. Set the necessary inputs for the returned object by using the `put()` method. The syntax of the method is:

   ```
   public void put(java.lang.String name, java.lang.String value)
   ```

   If the additional input is *not* present with the given `name`, then one is created. Otherwise, it is overwritten with the new `value`. It is recommended that large strings are not used either for `name` or for `value`.

Note: The `name` and `value` parameters must *not* contain = and the newline character (`\n`). The API behavior is undefined if `name` or `value` contain any of these characters.

Chapter 4
Managing Users

Creating a new user in RiskFort is a one-time operation, performed only when a new user is to be added to RiskFort database. Typically, this is an existing user of your application accessing RiskFort for the first time.

The Issuance Java API provides a programmable interface, which can be used by Java clients (such as Java Servlets and JSP pages) to send Issuance-related requests to RiskFort Server. This API creates a request message that is sent to the RiskFort Server, receives the response, and packages it as return response to be read by your application.

This chapter provides an overview of how to use the Java API to create users in RiskFort database and the methods and interfaces it implements. Next, a description of the interfaces and methods in this API used to read and update user details is given. Each user-related task description (create, read, or update user information) is followed by a sample code snippet that you can use to test the task. The chapter covers following topics:

Creating a User

To create a user in RiskFort database, you need to use the `User` and `UserRepositoryService` interfaces.

To create a user:

1. Ensure that you have initialized the Issuance API by using the `RiskFactory` class.

   See "Initializing the Issuance API" on page 3-28 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortissuanceAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-30 for more information on this.

3. Use the `UserManager.getNewUserObject(String userName, String groupName)` method to obtain an object that implements the `User` interface.

   This method returns a `User` object with username and the specified group.

4. Set the necessary properties for the returned object by using the `user.setProperty()` method.

5. Use the `UserRepManager.getGrpUserRepService()` method to obtain an object that implements the `UserRepositoryService` interface.

   This method returns an instance of the `UserRepositoryService` interface, which in turn returns the internal repository implementation for the specified group.

6. Finally, invoke the `create()` method of the `UserRepositoryService` interface to create the user.

   This method returns an instance of the `UserRepResult` class, which specifies the response code (`FAILURE`, `INVALID`, or `SUCCESS`), an optional message, and the transaction identifier generated at the RiskFort Server-end.

Handling Errors

Any errors that occurred during the execution of any of the Issuance API methods result in an `UserRepositoryException` exception being thrown. The `UserRepositoryException` object has `getCode` and `getTransactionId()` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for User Creation

Note: Refer to "Sample Code for User Operations" in Appendix B for a detailed working code sample.

You can use the following sample code snippet to understand how to implement the code for creating users in RiskFort Database.

```
public void createUser(User user, AdditionalInput additionalInput) throws
UserRepositoryException

{
```

```
    //Gets the implementation user repository service.
    UserRepositoryService srv = UserRepManager.getGrpUserRepService();

    //Get the implementation of User interface.
    User user = UserManager.getNewUserObject(userName, orgName);

    //set user properties
    user.setProperty(User.USER_PROPERTY_FIRSTNAME, firstName);
    user.setProperty(User.USER_PROPERTY_LASTNAME, lastName);
    user.setProperty(User.USER_PROPERTY_EMAILADDR, email);
    user.setPAM(pam);
    srv.create(user, additionalInput);
}
```

Reading User Details

To read the details of a user in RiskFort database, you need to use the `User` and `UserRepositoryService` interfaces.

To read user details:

1. Ensure that you have initialized the Issuance API by using the `RiskFactory` class.

   See "Initializing the Issuance API" on page 3-28 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortissuanceAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-30 for more information on this.

3. Use the `UserRepManager.getGrpUserRepService()` method to obtain an object that implements the `UserRepositoryService` interface.

   This method returns an instance of the `UserRepositoryService` interface, which returns the internal repository implementation for the specified groups.

4. Finally, invoke the `read()` method of the `User` interface to read the user details.

This method returns a fully-populated `User` object, and specifies whether the operation was a `FAILURE` or a `SUCCESS`.

Handling Errors

Any errors that occurred during the execution of any of the Issuance API methods result in an `UserRepositoryException` exception being thrown. The `UserRepositoryException` object has `getCode` and `getTransactionId()` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for Reading User Details

The following sample code snippet can be plugged in to your application code to read the details of a user from RiskFort database.

```
private User read (String userName, String orgName) throws
UserRepositoryException
{
    UserRepositoryService srv = UserRepManager.getGrpUserRepService();
    User user = srv.read(userName, orgName, additionalInput);
    return user;
}
```

Updating User Details

To update user information in RiskFort database, you need to use the `User` and `UserRepositoryService` interfaces and the `UserManager` class.

To update the details of a user:

1. Ensure that you have initialized the Issuance API by using the `RiskFactory` class.

   See "Initializing the Issuance API" on page 3-28 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortissuanceAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-30 for more information on this.

3. Use the `UserRepManager.getGrpUserRepService()` method to obtain an object that implements the `UserRepositoryService` interface.

   This method returns an instance of the `UserRepositoryService` interface, which returns the internal repository implementation for the specified group.

4. Use the `UserManager.getUserDetails(userName, groupName)` method to obtain an object that implements the `User` interface.

   This method returns a `User` object with username and the specified group.

5. Check and set the necessary properties for the returned object by using the `user.setProperty()` method.

6. Finally, invoke the `update()` method of the `UserRepositoryService` interface to update the user details.

   This method returns an instance of the `UserRepResult` class, which specifies the response code (`FAILURE`, `INVALID`, or `SUCCESS`), an optional message, and the transaction identifier generated at the RiskFort Server-end.

Handling Errors

Any errors that occurred during the execution of any of the Issuance API methods result in an `UserRepositoryException` exception being thrown. The `UserRepositoryException` object has `getCode` and `getTransactionId()` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for Updating User Details

The following sample code snippet can be plugged in to your application code to update the details of a user in RiskFort database.

```
private String updateUser(User user, AdditionalInput additionalInput)
{
    UserRepositoryService srv = UserRepManager.getGrpUserRepService();
    String errorCode = null;
    User user;
    try {
              user = UserManager.read(userName, orgName, additionalInput);
          }
    catch (UserRepositoryException e1)
```

```
        {
             errorCode = e1.getCode();
             return errorCode;
        }
    if (fname != null)
             user.setProperty(User.USER_PROPERTY_FIRSTNAME, fname);
    if (lname != null)
             user.setProperty(User.USER_PROPERTY_LASTNAME, lname);
    if (email != null)
             user.setProperty(User.USER_PROPERTY_EMAILADDR, email);
    if (pam != null)
             user.setPAM(pam);

    try
       {
             srv.update(user, additionalInput);
        }
    catch (UserRepositoryException e) {
             errorCode = e.getCode();
        }
    return errorCode;
}
```

Chapter 5
Collecting Device ID and Machine FingerPrint

RiskFort uses user-device (desktop computers, laptops, and notebooks) information as one of the parameters to determine the risk associated with a login attempt or a transaction. As a result, the verification of the online identity of the end user is a challenge. RiskFort also uses Device ID and MFP technologies (in addition to other inputs, as discussed in Chapter 2, "Understanding RiskFort Workflows") for this purpose. These technologies enable RiskFort to build the user profile and to transparently provide accurate results by using the hardware that users already possess, without changing the end-user experience significantly.

This chapter provides detailed information on how to get and set Device ID and collect the MFP data from the end user's device and pass it to RiskFort. It covers the following topics:

Device ID and MFP Basics

This section introduces you to the Device ID and MFP basics.

Device ID

The *Device ID* is a device identifier string that RiskFort generates and stores as a cookie on the end user's system to identify and track the device that the end user uses for logging into your online application and perform transactions. The information is in encrypted format.

The RiskFort Device ID can be stored as:

- **Flash cookie:** Flash Shared Object (FSO) stored with `.sol` or `.ssl` (if SSL is being used for communication) extension. Typically, on Windows this cookie is available in the Flash Player directory of the user's profile:

`<user_profile>\Application Data\Macromedia\FlashPlayer\#SharedObjects\`

In the preceding path, `<user_profile>` typically represents `C:\Documents and Settings\<user_name>\`.

- **Browser cookie:** HTTP cookie whose extension and storage location depends on the browser used by the end user. Typically, on Windows this cookie is available in:
  - **Microsoft IE:**

    `C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files\`

  - **Mozilla Firefox:**

    `C:\Documents and Settings\<user_name>\Application Data\Mozilla\Firefox\Profiles\<value>.default\cookies.sqlite`

  - **Safari:**

    `C:\Documents and Settings\<user_name>\Application Data\Apple Computer\Safari\cookies.plist`

Why Flash Cookies are Recommended

Most Browser cookies have an expiry period or are automatically deleted after a certain period of time when there is no activity. In addition, Browser cookies can also be accidentally deleted or cleared by the user. In such cases, RiskFort will generate the `INCREASEAUTH` advice. As a result, the user must undergo secondary authentication, despite the fact that their device information might exist in the RiskFort database.

Unlike Browser cookies, Flash cookies do not have an expiry period and persist unless manually deleted. Also because an average user cannot easily locate and delete these FSOs from the browser interface, the chances of Flash cookies being accidentally deleted by the user are much lower than the Browser cookies.

In addition, the Flash solution provided by Arcot enables you to handle the cross-domain attacks better than the Browser cookies.

MFP

Machine fingerprinting (also referred to as Device fingerprinting or PC fingerprinting) is a device identification technique used for gathering user's system information, such as browser, operating system, and other attributes. RiskFort then analyzes this information to generate a risk profile of a device in real-time.

Some of the MFP attributes collected by RiskFort from the end user's device include:

- Operating system name and version
- Browser information (such as name, major version, minor version, JavaScript version, HTTP headers, User Agent)
- Screen settings (such as height, width, color depth)
- System information (such as time zone, language, system locale)

For every transaction by the end user, RiskFort matches the corresponding MFP stored in its database with the incoming MFP. If this match percentage (%) is less than the value specified in the Miscellaneous Rules Configuration screen of the Administration Console, then the transaction is considered risky.

Files that You Will Need

You will need the files listed in Table 5-1, available when you install RiskFort, to collect the Device ID and MFP information from the end user's device.

Table 5-1. Files Required for Collecting Device ID and Machine FingerPrint (MFP) Information

Location	File Name	Applicable for Cookie Type	Description

Windows:
`<install_location>\Arcot Systems\sdk\javascript\`
 	
Solaris:
`<install_location>/arcot/sdk/javascript/`
	`rfutil.js`	Flash	This file provides JavaScript functions to get and set the Flash cookies for Device ID.

		Browser	This file provides JavaScript functions to `get` and `set` HTTP cookies.

Table 5-1. Files Required for Collecting Device ID and Machine FingerPrint (MFP) Information

Location	File Name	Applicable for Cookie Type	Description
Windows: <*install_location*>\Arcot Systems**sdk****javascript**\ **Solaris:** <*install_location*>/arcot/**sdk**/**javascript**/	mfp_json.js	Flash Browser	This file contains JavaScript functions to gather the MFP-related information from the end user's device and generate the single encoded String with all the machine fingerprint values. Your application must include this file and call its `jsonSignature()` method at the client end. At your application-end, you must call the `buildDeviceSignature()` method to convert it to a format that RiskFort can understand. You must do this *before* you call the `evaluateRisk()` method. This method should be called from the page, such as the index or login page, for which your online application requires a risk assessment.

Table 5-1. Files Required for Collecting Device ID and Machine FingerPrint (MFP) Information

Location	File Name	Applicable for Cookie Type	Description

Windows:
`<install_location>\Arcot Systems\sdk\flash\`
 	
Solaris:
`<install_location>/arcot/sdk/flash/`
	`rfdevice.swf`	Flash	This file contains the code to mange Flash cookies and provide support for cross-domain access to the cookies, so that your application pages from one domain can access the cookies set in a different domain.
			
			Important: In all your online application pages, you must refer to `rfdevice.swf` with complete and the same URL.

Windows:
`<install_location>\Arcot Systems\sdk\flash\`
 	
Solaris:
`<install_location>/arcot/sdk/flash/`
	`crossdomain.txt`	Flash	This file specifies the list of domains that are allowed to access the Flash cookie. (By default, only sub-domains of the domain from where the RiskFort Flash movie is served are allowed to access the Flash cookie.) The format of a domain entry in `crossdomain.txt` is:
			
			`&domainName=<pipe-separated_domain_list>&`
			
			Important: This file must reside in the same location as `rfdevice.swf`.

Configuring Device ID

As discussed earlier in this chapter, the Device ID is an important aspect of the verification of the device being used by the end user for the current transaction. Therefore, a Flash cookie or a browser (HTTP) cookie needs to be set on the user computer. This section discusses the how to configure for the required cookie type to be set on the end-user device:

- Configuring Flash Cookies
- Configuring HTTP Cookies

Configuring Flash Cookies

To configure for a Flash cookie to be set on the end-user computer, you need the following files:

- `rfutil.js`
- `rfdevice.swf`
- `crossdomain.txt`

Perform the following tasks to configure a Flash cookie:

1. Include `rfutil.js` in your application page(s) that `get` or `set` the Flash cookies.
   a. Copy `rfutil.js` to an appropriate Web application folder (say, *$APP_SERVER_HOME*/**javascript**/) that is relative to the location where the page (in which you are including the `rfutil.js` file) is available.
   b. Include the following JavaScript code in the relevant Web page of your application:

```
<script type="text/javascript" src="location_to_rfutil.js/"></script>
```

   In the preceding code snippet, replace *location_to_rfutil.js* with the relative path to `rfutil.js`.
2. Copy `rfdevice.swf` and `crossdomain.txt` to the appropriate Web application folder (say, *$APP_SERVER_HOME*/**javascript**/).

Important: The `crossdomain.txt` file *must* reside in the same location as `rfdevice.swf`.

3. If your Flash cookie will be accessed across domains, then add the list of domains in the following format in `crossdomain.txt`:

```
&domainName=<pipe-separated_domain_list>&
```

   For example, for a Web site that has aggregated pages from its own site and its partner site, the entry will be as follows:

```
&domainName=*.my-bank.com|*.my-partner.com&
```

4. Ensure that in the all application pages, `rfdevice.swf` is referred with absolute and same URL.

   For example, if `rfdevice.swf` is delivered from *login.my-bank.com*, then both sites (*login.my-bank.com* and *online.my-partner.com*) must include `rfdevice.swf` from *login.my-bank.com*.

Configuring HTTP Cookies

To configure for an HTTP cookie to be set on the end-user computer, you must include `rfutil.js` in your application page(s) that get or set the HTTP cookies.

To do so:

1. Copy `rfutil.js` to an appropriate Web application folder that is relative to the location where the page (in which you are including the `rfutil.js` file) is available.
2. Include the following JavaScript code in the relevant Web page of your application:

```
<script type="text/javascript"
src="<location_to_rfutil/>.js"></script>
```

   In the preceding code snippet, replace <*location_to_rfutil*>.js with the relative path to `rfutil.js`.

Building the MFP and Collecting the Device ID

To implement the functionality of the MFP and Device ID collection, you must implement corresponding code snippets into each page of your application that contains an event that requires risk assessment. For example, for risk assessment of a login event, your application must implement the required JavaScript files and code snippets into the login page. Similarly for a pre-login event, the steps discussed in this section must trigger when a user accesses the first page of your online application.

The steps to build the MFP and collect the Device ID from the end user's device are:

- Step 1: Include the Javascript Files
- Step 2: Collect the MFP
- Step 3: Reformat the MFP
- Step 4: Collect the IP Address
- Step 5: Collect the Device ID

You can implement these steps either in a single page of your online application, or across multiple pages (depending on how many pages you show during the login process) *before* you call the `evaluateRisk()` method.

Step 1: Include the Javascript Files

You will need to modify the appropriate Web pages, such as the login or index page (say, `index.jsp` or `login.jsp`) to enable them to gather MFP-related information and collect the Device ID (cookie) from the end user's computer.

Note: See "Enrollment Workflows" on page 2-7 for more information on when and how RiskFort sets the Device ID on end user's system.

To implement the script codes:

1. Include the `mfp_json.js` Javascript code in the required application page(s):

```
<script type="text/javascript" src="javascript/mfp_json.js"></script>
```

In the preceding code snippet, we assume that `mfp_json.js` is located in a folder that is relative to the folder containing `index.jsp`.

2. Include the `rfutil.js` Javascript code in the required application page(s) for getting the Device ID set on the end user's system.

```
<script type="text/javascript" src="javascript/rfutil.js"></script>
```

3. Copy the `mfp_json.js` file *from* the following location to the appropriate Web application folder (*$APP_SERVER_HOME*/**javascript**/):
   - **On Windows**

     <install_location>\Arcot Systems**sdk****javascript**\
   - **On UNIX Platforms**

     <install_location>/arcot/**sdk**/**javascript**/
4. Copy the `rfutil.js` file *from* the following location to the appropriate Web application folder (*$APP_SERVER_HOME*/**javascript**/):
   - **On Windows**

     <install_location>\Arcot Systems**sdk****javascript**\
   - **On UNIX Platforms**

     $ARCOT_HOME/arcot/**sdk**/**javascript**/

Step 2: Collect the MFP

Implement the following for MFP collection:

Note: Refer to the code in the "Sample Code Reference" on page 5-56 section to understand these steps better.

1. Include the following in your HTML code *before* processing anything related to MFP:

```
  <body onload="init()">


<form name="CollectMFPToEvaluate" method="POST">
```

```
     <input type="hidden" name="MFP" length=1056>
     <input type="hidden" name="IpAddress">
     <input type="hidden" name="CallerID">
     <input type="hidden" name="DeviceID">
     <input type="button" value="Login"
onClick="callfuncAfterInitFinished();">
     </form>


  </body>
  </html>
```

2. Call the `jsonSignature()` method, which is a part of the `mfp_json.js` Javascript file that you included in Step 1: Include the Javascript Files:

```
   document.CollectMFPToEvaluate.MFP.value = jsonSignature();
```

This method gathers the MFP-related information from the end user's device and returns a JSON string.

Sample Application Reference

You can also refer to `ArRFMFPCollectionServlet.java`, which is a part of the RiskFort Sample Application. This file showcases the collection of MFP and other required information and sets these parameters for the session. After you deploy the Sample Application, this file is available at:

<RISKFORT_SAMPLEAPP_HOME>\WEB-INF\classes\com\arcot**riskfort****sampleapp****servlets**\

Step 3: Reformat the MFP

The MFP received by your online application must now be reformatted before you can send it to RiskFort Server. This is because the signature that you collected in the previous step (Step 2: Collect the MFP) is browser-specific, and varies according to the browser that the end user is using. Therefore, you must convert this MFP to a RiskFort native string that the RiskFort Server can understand and process.

Use the following method in the `com.arcot.riskfortAPI.DeviceContext` API to reformat the collected MFP to a JSON string:

```
buildDeviceSignature(signatureJSON, httpHeaders, extraInfo)
```

Sample Application Reference

You can also refer to `ArRFEvaluateHelper.java`, which is a part of the RiskFort Sample Application. This file showcases the `buildDeviceSignature()` method and calls the required API for processing. This file is available at the following location:

<RISKFORT_SAMPLEAPP_HOME>\WEB-INF\classes\com\arcot**riskfort\sampleapp\helpers**\

Step 4: Collect the IP Address

RiskFort does not provide any mechanism to collect the IP address of the end-user device. As a result, you must implement your own logic to do so.

See section, "Collecting the IP Address" on page 5-61 for recommendations.

Step 5: Collect the Device ID

After the Device ID (cookie) is set on the end user's system (as discussed in "Configuring Device ID" on page 5-48) you must now ensure that you `get` the cookie along with the MFP. However, this collection mechanism depends on the type of cookie set on the user's system.

In Case of Flash Cookies

If you are using Flash cookies, then:

1. Ensure that the `init()` is the *first* method in the `<body>` element of the Web page:

   ```
   <body onload="init();">
   ```

2. Ensure that you have defined the `init()` function. For example, you can use the following code snippet:

   ```
   function init()
    {
   ```

```
   var so = new
SWFObject("<%=request.getContextPath()%>/flash/rfdevice.swf",
            "cookiemanager", "0", "0", "6", "#ffff00");
   so.addParam("allowScriptAccess", "always");
   so.write("flashcontent");

   var divid = document.getElementById("flashsetting");
   divid.style.display = 'block';
 }
```

Note: If you are already using an `init()` function, then ensure that you copy only the required code to the function definition.

3. Ensure that on click of the **Login** (or **Submit**) button on the page, the following code snippet is called:

```
    <input type="button" value="Login"
onClick="callfuncAfterInitFinished();">
```

4. Ensure that you have defined the `callfuncAfterInitFinished()` function. This function is required to make the next statement in the code wait until the Flash movie is loaded completely. Otherwise, the subsequent read or write operations on the Flash cookie might fail.

   For example, you can use the following code snippet:

```
function callfuncAfterInitFinished()
  {
    /*
     Once movie loading is complete, the arcotIsInitDone variable is
     set to 1. arcotMaxRetries is the maximum number of times the
     check can be performed.
     This is defined in rfutil.js.
    */
```

```
    if (arcotIsInitDone != 1 && arcotMovieLoadCheckCnt <
arcotMaxRetries)
      {
         arcotMovieLoadCheckCnt++;
         /*
           ARCOT_RETRY_PERIOD_MSEC is the time interval after which
           the check is done again. This is defined in rfutil.js.
         */
         setTimeout("callfuncAfterInitFinished()",
ARCOT_RETRY_PERIOD_MSEC);
      }
    //If the flash movie has loaded completely.
    else if (arcotIsInitDone == 1)
      {
         //You must ensure that you call the collectSystemInfo()
         //function from the body of the login() method.
         login();
      }
    //If movie-loading has not happened within reasonable time.
    else
      {
         //You must ensure that you call the collectSystemInfo()
         //function from the body of the login() method.
         login();
      }
  }
```

In Case of HTTP Cookies

In case of Browser (HTTP) cookies, you *do not need to call* the `init()` method, as is the case with Flash cookies. However, you must ensure that:

1. Ensure that on click of the **Login** (or **Submit**) button on the page, the following code snippet is called:

```
    <input type="button" value="Login"
onClick="collectSystemInfo();">
```

2. Ensure that you have defined the `collectSystemInfo()` function. For example, you can use the following code snippet:

```
function collectSystemInfo()
{
  //Set the cookie type as per your requirements.
    {<code for cookie type>}

  // jsonSignature() collectes the machine fingerprint.
     document.CollectMFPToEvaluate.MFP.value = jsonSignature();

     document.CollectMFPToEvaluate.submit();
  }
```

After you have built the MFP and collected the Device ID, as required, they must be passed as inputs to `evaluateRisk()` method. See "Performing Risk Evaluation" on page 6-63 for more information.

Sample Code Reference

The following sample code illustrates the modifications required for implementing RiskFort's MFP and Device ID collection mechanism. It showcases the collection logic in one file (say, **`index.jsp`**). However, you can implement appropriate code snippets in different pages, depending on the number of pages you show *before* you call the `evaluateRisk()` method.

```
---------------------------------------------------------------------
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">



<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">


//mfp_json.js collects the MFP from the client machine.
<script type="text/javascript"
src="<%=request.getContextPath()%>/javascript/mfp_json.js"></script>


//rfutil.js sets and gets HTTP/Flash cookies and Flash functions.
<script type="text/javascript"
src="<%=request.getContextPath()%>/javascript/rfutil.js"></script>


<script language="javascript">



  function init()
  {


    var so = new SWFObject("<%=request.getContextPath()%>/flash/rfdevice.swf",
"cookiemanager", "0", "0", "6", "#ffff00");


    so.addParam("allowScriptAccess", "always");
    so.write("flashcontent");


    var divid = document.getElementById("flashsetting");


    //intially hidden
    divid.style.display = 'block';
  }
```

```
function callfuncAfterInitFinished()
{

  /*
   Once movie loading is complete, the arcotIsInitDone variable is set to 1.
   arcotMaxRetries is the maximum number of times the check can be performed.
   This is defined in rfutil.js.
  */

  if (arcotIsInitDone != 1 && arcotMovieLoadCheckCnt < arcotMaxRetries)
    {
       arcotMovieLoadCheckCnt++;

       /*
        ARCOT_RETRY_PERIOD_MSEC is the time interval after which the check is
         done again. This is defined in rfutil.js.
       */

       setTimeout("callfuncAfterInitFinished()", ARCOT_RETRY_PERIOD_MSEC);

    }

  //If the flash movie has loaded completely.
  else if (arcotIsInitDone == 1)
    {
       login();
    }

  //If movie-loading has not happened within reasonable time.
  else
    {
```

```
         login();
      }
  }


  function collectSystemInfo()
  {
        // Set the value of the var as needed.
        var cookietype = "flashcookie";

        if(cookietype == "flashcookie")
          {
             document.CollectMFPToEvaluate.DeviceID.value =
getFlashCookie"CookieName");
          }
          else{
             document.CollectMFPToEvaluate.DeviceID.value =
getBrowserCookie("CookieName");
          }

        // jsonSignature() collectes the machine fingerprint.
        document.CollectMFPToEvaluate.MFP.value = jsonSignature();

        document.CollectMFPToEvaluate.submit();
  }


  function login()

  {

   collectSystemInfo();
```

```
 // Process form for submission.
    //........
    //........
    //........
  }



</script>
</head>
<!--
Arcot Form to get user informatiom.Have this always before processing the form
for user login.
-->


<body onload="init()">



  <form name="CollectMFPToEvaluate" method="POST">
     <input type="hidden" name="MFP" length=1056>
     <input type="hidden" name="IpAddress">
     <input type="hidden" name="CallerID">
     <input type="hidden" name="DeviceID">
     <input type="button" value="Login" onClick="callfuncAfterInitFinished();">
     </form>


</body>
</html>
```

Collecting the IP Address

The end user accessing your online application might be a home user or might be accessing it from their corporate network. In case of latter category of users, chances are that they might be "hidden" behind a proxy server. As a result, the way you will collect the IP address of an end user who is accessing your online application from behind a proxy will be different from the user who accesses it directly from home.

If the End User is Accessing Your Application Directly

If the end user is accessing your application directly, then you can use the `getRemoteAddr()` method of the `HttpServletRequest` interface in your JSP. This method returns a string that contains the IP address of the client that sent the request.

Chapter 6
Performing Risk Evaluation

When a user accesses your online application, the application forwards the request to RiskFort for risk analysis. RiskFort can evaluate risk for all users, whether they are first-time users (and therefore not "known" to RiskFort) or if they are already enrolled with the RiskFort system.

The Risk Evaluation Java API provides a programmable interface, which can be used by Java clients (such as Java Servlets and JSP pages) to send risk evaluation-related requests to RiskFort Server. This API creates a request message that is sent to the RiskFort Server, receives the response, and packages it as return structures to be read by the client.

This chapter provides an overview of how to use the Java API to perform risk evaluation and the methods and interfaces it implements. Next a description of the interfaces and methods in the API that you can use to list and delete associations in RiskFort database is given. Each risk evaluation-related and association management task description is followed by a sample code snippet that you can use in your code to perform the task. The chapter covers following topics:

Evaluating Risks and Performing Post-Evaluation

To evaluate the risk associated with a transaction and to perform the subsequent post-evaluation, you need to use the `RiskXActionAPI` interface (in the `com.arcot.riskfortAPI` package. This interface represents the client-side interface to RiskFort Server's risk evaluation functionality and exposes the APIs supported for various workflows for risk evaluation.

To evaluate risk of a transaction and perform post-evaluation tasks:

1. Ensure that you have initialized the Risk Evaluation API by using the `RiskFactory` class.

   See "Initializing the Risk Evaluation API" on page 3-33 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-35 for more information on this.

3. Use the `RiskFactory.getRiskXActionAPI()` static method to obtain an object that implements the `RiskXActionAPI` interface.

   This method returns a usable instance of the `RiskFactory` class that implements the `RiskXActionAPI` interface.

4. Use the `RiskXActionAPI.evaluateRisk()` method to obtain an object of the `RiskAssessment` class.

   This method requires the following input parameters:

   a. Define and set the `CallerID` string variable, which will be used by your application for tracking purposes across calls.

   b. Use the `DeviceContext()` method to obtain an object of the `DeviceContext` class and build the JSON Signature by using the `DeviceContext.buildDeviceSignature()` method.

   c. Call the JSON Signature that you built, as discussed in Chapter 5, "Collecting Device ID and Machine FingerPrint".

   d. Use the `LocationContext()` method to obtain an object of the `LocationContext` class and set the necessary properties for the returned object by using the `LocationContext.setIpAddress()` method.

   e. Use the `UserContext()` method to obtain an object of the `UserContext` class and set the necessary properties for the returned object by using the `UserContext.setUserId()` method.

   f. Use the `TransactionContext()` method to obtain an object of the `TransactionContext` class and, if required, set the necessary properties for the returned object by using the methods in this class (`setAction()` and `setChannel()`).

   g. If you are using extra information, then use the `AdditionalInputs()` method to obtain an object of the `AdditionalInputs` class and set the necessary properties for the returned object by using the `AdditionalInputs.put()` method.

      These are additional inputs in form of `name-value` pairs. For example:

      ```
      MerchantID=id;MerchantCountry=country;MerchantName=name
      ```

5. Obtain the resulting `RiskAssessment RiskAdvice` object by using the `RiskAssessment.getRiskAdvice()` method.

Important: If the advice is `INCREASEAUTH`, then your application must perform secondary authentication and pass the result of this authentication to RiskFort by using the `PostEvaluate()` method.

6. Finally, use the `RiskXActionAPI.postEvaluate()` method to determine the final outcome of the transaction and return a `PostEvaluateResponse` object.

   You will need to pass the result of secondary authentication to the method, if you performed any.

7. You can query the resulting `PostEvaluateResponse` object by using the `PostEvaluateResponse.isAllowAdvised()` method to determine whether or not to permit the transaction to proceed.

Handling Errors

Any errors that occurred during the execution of any of the Risk Evaluation API methods result in an `RiskException` exception being thrown. The `RiskException` object has `getErrorCode` and `getTransactionId` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for Risk Evaluation and Post Evaluation

Note: Refer to "Sample Code for Risk Evaluation and Post-Evaluation" in Appendix B for a detailed working code sample.

You can use the following sample code snippet to understand how to implement the risk evaluation and post-evaluation capability of RiskFort in your application code.

```
public static void sampleCode() {
        String propertyLocation=
"/properties/riskfort.risk-evaluation.properties";
        try {
             RiskFactory.initialize(propertyLocation);
             RiskXActionAPI riskXActionAPI = RiskFactory.getRiskXActionAPI();
             String callerId;
             UserContext userContext = new UserContext();
             LocationContext locationContext = new LocationContext();
             DeviceContext deviceContext = new DeviceContext();
             TransactionContext transactionContext = new TransactionContext();
             AdditionalInputs additionalInputs = new AdditionalInputs();
```

```
            // string used by the calling application for tracking across
             // calls
            callerId="MyApplicationTrackingId";

            // Unique identifier for the user. In case of a Bank it may be
             // user's bank account number
            // It may be name of the user in some other case.
            userContext.setUserId("USER1");

            // IP address of the user's machine, typically, extracted from
             // the HTTP header
            locationContext.setIpAddress(InetAddress.getByName("10.150.1.1"));

            // JSON Signature comes from mfp_json.js, in this example the
             // signature is hard coded
            // for the sample use.
            String jsonSignature =
"{\"navigator\":{\"platform\":\"Win32\",\"appName\":\"Netscape\",\"appCodeName
\":\"Mozilla\",\"appVersion\":\"5.0 (Windows;
en-US)\",\"language\":\"en-US\",\"oscpu\":\"Windows NT
5.0\",\"vendor\":\"\",\"vendorSub\":\"\",\"product\":\"Gecko\",\"productSub\":
\"20070312\",\"securityPolicy\":\"\",\"userAgent\":\"Mozilla/5.0 (Windows; U;
Windows NT 5.0; en-US; rv:1.8.0.11) Gecko/20070312
Firefox/1.5.0.11\",\"cookieEnabled\":true,\"onLine\":true},\"plugins\":[{\"nam
e\":\"Adobe Acrobat Plugin\",\"version\":\"7.00\"},{\"name\":\"Macromedia
Director\",\"version\":\"10.1\"},{\"name\":\"Windows Media Player Plug-in
Dynamic Link Library\",\"version\":\"\"},{\"name\":\"Macromedia Shockwave
Flash\",\"version\":\"9.0\"},{\"name\":\"Java Virtual
Machine\",\"version\":\"1.6.0\"}],\"screen\":{\"availHeight\":690,\"availWidth
\":1024,\"colorDepth\":32,\"height\":768,\"pixelDepth\":32,\"width\":1024},\"e
xtra\":{\"javascript_ver\":\"1.6\",\"timezone\":-330}}";
            deviceContext.buildDeviceSignature(jsonSignature,null,null);
            String
userDeviceId="GPXp+4e0hzzxzh6YLlPZqKgXCGbBXB8E0ghZnFXHq8o3HLRaww6c4g==";
            // The device id collected from the user machine
            deviceContext.setDeviceID("HTTP_COOKIE", userDeviceId);
```

```
            // Providing the addition inputs.
            additionalInputs.put("MerchantID","id") ;
            additionalInputs.put("MerchantCountry","country") ;
            additionalInputs.put("MerchantName","name") ;

            transactionContext.setAction("Login");
            RiskAssessment riskAssessment=null;
            riskAssessment = riskXActionAPI.evaluateRisk(callerId ,
deviceContext, locationContext , userContext, transactionContext,
additionalInputs);
            boolean secondaryAuthenticationStatus = true;
            String associationName = "USER1inHomePC";

            if
(riskAssessment.getRiskAdvice().equals(RiskAssessment.RISK_ADVICE_INCREASEAUTH
)) {
                        // then you may ask for secondary authentication
                        //if( secondaryAuthentication succeeded )
                        //          secondaryAuthenticationStatus = true;
                        //else
                        //          secondaryAuthenticationStatus = false
            }

            PostEvaluateResponse postEvaluateResponse =
                        riskXActionAPI.postEvaluate(callerId, riskAssessment,

secondaryAuthenticationStatus, associationName);
            if( postEvaluateResponse.isAllowAdvised() ) {
                        //Allow the transaction to be completed
            }
            else {
                            //Deny and terminate the transaction
            }
```

```
            } catch (IOException e) {
           //Looks like the property file location is not valid
               e.printStackTrace();
            } catch (RiskException e) {
               //One of the RiskFort API calls broke
               e.printStackTrace();
            }
}
```

Managing Associations

RiskFort uniquely identifies a user as a valid user of your system by automatically associating (or binding) a user to the device that they use to access your application. This is referred to as an *association* (or device binding) in RiskFort terminology. Users who are not bound are more likely to be challenged in order to be authenticated.

RiskFort also allows users to be bound to more than one devices. For example, a user can use a work and a home computer to access your application. Similarly, you can bind a single device to more than one users. For example, members of a family can use one computer to access your application.

Important: Arcot strongly recommends that you discourage users from creating associations with publicly shared devices, such as systems in an Internet cafe or kiosk.

Association management includes:

- Listing Associations
- Deleting Associations

Listing Associations

To list all the stored associations for a specified user:

1. Ensure that you have initialized the Risk Evaluation API by using the `RiskFactory` class.

   See "Initializing the Issuance API" on page 3-28 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-35 for more information on this.

3. Use the `RiskFactory.getRiskXActionAPI()` static method to obtain an object that implements the `RiskXActionAPI` interface.

   This method returns a usable instance of the `RiskFactory` class that implements the `RiskXActionAPI` interface.

4. Define and set the `CallerID` string variable, which will be used by your application for tracking purposes across calls.

5. Use the `UserContext()` method to obtain an object of the `UserContext` class.

6. Set the necessary properties for the returned object.

   For example, you can set the user ID by calling the `UserContext.setUserId()` method.

7. Call the `RiskXActionAPI.listAssociations()` method that returns a `ListAssociationResponse` object.

   The `ListAssociationResponse.getAllAssociations()` method returns an array of all known associations for the specified user.

Handling Errors

Any errors that occurred during the execution of any of the Risk Evaluation API methods result in an `RiskException` exception being thrown. The `RiskException` object has `getErrorCode` and `getTransactionId` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for Listing Associations

The following code snippet can be plugged in to your application code to list all existing associations.

```
public ListAssociationResponse listAssociations(java.lang.String callerId,
         UserContext userContext,
         AdditionalInputs additionalInputs)
         throws RiskException
```

Deleting Associations

To delete the specified user-device association for a user:

1. Ensure that you have initialized the Risk Evaluation API by using the `RiskFactory` class.

   See "Initializing the Risk Evaluation API" on page 3-33 for more information on this.

2. If required, prepare the additional inputs for the transaction, by using the `AdditionalInputs` class in the `com.arcot.riskfortAPI.AdditionalInputs` package.

   See "Preparing Additional Inputs" on page 3-35 for more information on this.

3. Use the `RiskFactory.getRiskXActionAPI()` static method to obtain an object that implements the `RiskXActionAPI` interface.

   This method returns a usable instance of the `RiskFactory` class that implements the `RiskXActionAPI` interface.

4. Define and set the `CallerID` string variable, which will be used by your application for tracking purposes across calls.

5. Use the `UserContext()` method to obtain an object of the `UserContext` class.

6. Set the necessary properties for the returned object.

   For example, you can set the user ID by calling the `UserContext.setUserId()` method.

7. Obtain the name of the association that you want to delete by using the `ListAssociationResponse` array returned by the `RiskXActionAPI.listAssociations()` method.

8. Call the `RiskXActionAPI.deleteAssociation()` method that returns a `DeleteAssociationResponse` object.

   The `DeleteAssociationResponse.DeleteAssociationResponse()` method deletes the association for the user that you specified.

Handling Errors

Any errors that occurred during the execution of any of the Risk Evaluation API methods result in an `RiskException` exception being thrown. The `RiskException` object has `getErrorCode` and `getTransactionId` members that contain the error code and transaction identifier (generated at the RiskFort Server-end) associated with the error that occurred.

Sample Code for Deleting Associations

The following code snippet can be plugged in to your application code to delete a user-device association.

```
public DeleteAssociationResponse deleteAssociation(java.lang.String callerId,
       UserContext userContext,
       java.lang.String associationName,
       AdditionalInputs additionalInputs)
     throws RiskException
```

Appendix A
Additional SDK Configurations

This appendix discusses the following topics:

Configuring Multiple RiskFort Server Instances

To configure the Java SDKs to communicate with multiple RiskFort Server instances, you must edit the properties file.

By default, these file provide the entries to configure *one* RiskFort Server instance. These entries are appended with `1`, which indicates that only one instance is configured. Depending on the number of instances you want to configure, you must duplicate these entries and append the instance number accordingly.

To configure multiple server instances:

1. Depending on the SDK you are configuring, open the respective properties file available in the following folder:
   - **Windows:**

     `<install_location>\Arcot Systems\sdk\java\properties\`
   - **Unix-Based Platforms:**

     `<install_location>/arcot/sdk/java/properties/`
2. Set the value of the `transport.<n>` parameter to the required communication mode. By default, it is set to `TCP`.

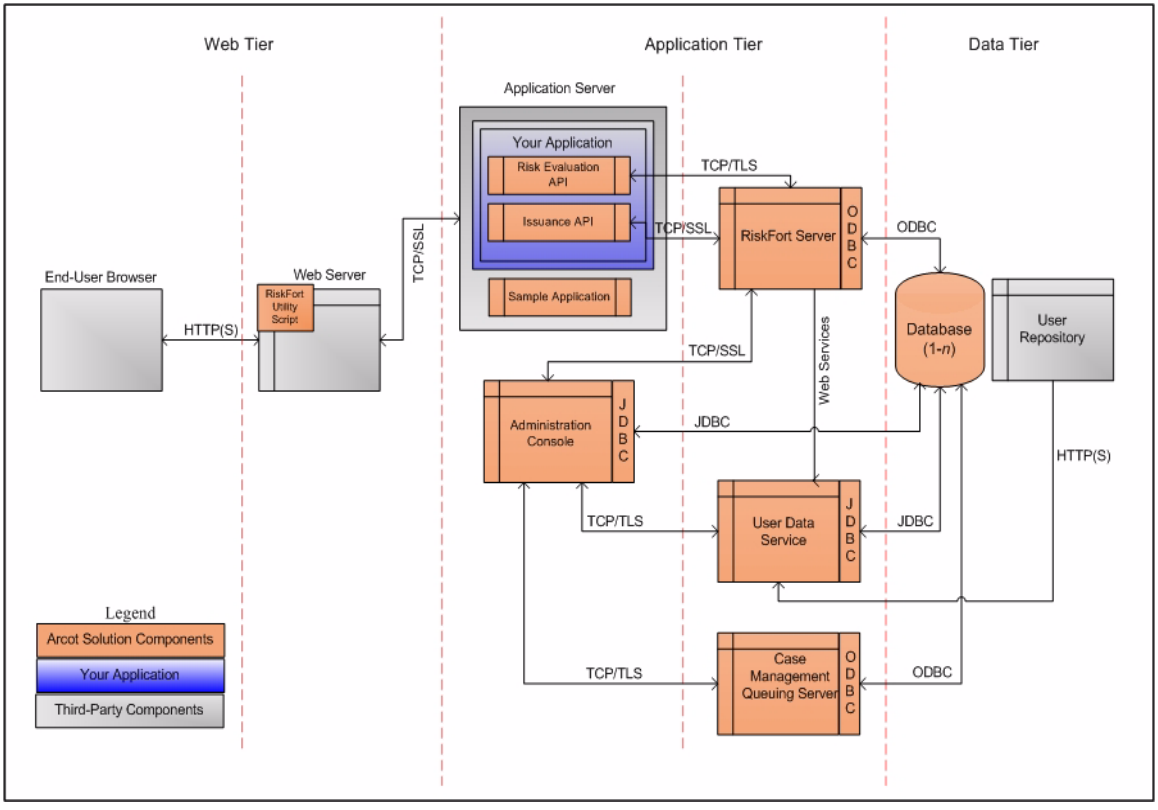
   See "Setting Up SSL Communication Between SDKs and RiskFort Server" on page A-74 if you want to change the communication mode.
3. Set the value of `host.<n>` parameter to the host name or the IP address of the required server instance.

4. Set the value of `port.<n>` parameter to the port number on which the **RiskFort Native** or the **Transaction Web Service** protocol is listening.
5. Save the changes and close the file.

Setting Up SSL Communication Between SDKs and RiskFort Server

In addition to supporting TCP-based communication between RiskFort Server and the SDKs, RiskFort also supports Secure Socket Layer (SSL) for secure communication between these components. RiskFort can be configured for one-way Secure Socket Layer (SSL) with server-side certificates or two-way SSL with server-side and client-side certificates between the Server and SDKs, as shown in Figure A-1.

Figure A-1 Communication Modes



One-Way SSL Communication

To ensure integrity and confidentiality of the data being exchanged during a transaction, RiskFort Server supports one-way SSL communication with:

- RiskFort SDKs
- Sample Application
- `arrrfadmin` tool

For each transaction, RiskFort Server presents its signed certificate to Risk Evaluation and Issuance APIs for verification. RiskFort APIs then verify the identity of RiskFort Server and the authentication is completed.

Two-Way SSL Communication

To ensure integrity and confidentiality of the data being exchanged during a transaction, RiskFort Server supports two-way SSL communication (in addition to the one-way SSL communication) with:

- **Evaluation Callout**

Note: RiskFort enables you to write your own custom Evaluation rule, based on your business requirements. This custom rule is called Evaluation Callout.

Refer to *Arcot RiskFort 2.2.6 Administration Guide* for more information on this.

- **Scoring Callout**

Note: RiskFort also enables you to write your own custom Scoring logic called Scoring Callout.

Refer to *Arcot RiskFort 2.2.6 Administration Guide* for more information on this.

If you are using an Evaluation Callout or a Scoring Callout, then your Callout must present its signed certificate to the RiskFort Server, which verifies the identity of the Callout and in turn presents its signed certificate to the Callout for verification. If the Callout verifies the identity of RiskFort Server successfully, the authentication is completed.

Configuring the SSL Communication

To secure the communication between RiskFort Server and Java SDKs:

1. *Before* you enable the SSL communication between Java SDKs and RiskFort Server, you must:
   a. Obtain a digital certificate from a trusted Certificate Authority.
   b. Expose your application over an HTTPS-enabled server port.
2. Log in as a Master Administrator (MA).
3. Using the **Protocol Configuration** page of the RiskFort Administration Console:
   a. Set the Transport Security mode for **RiskFort Native** protocol to **SSL**.
   b. Specify the root certificate of the trusted CA for **Client Store**.
   c. Browse to the required **Certificate Chain** and **Private Key**.
   d. Save the changes.
4. Navigate to the following location:
   - **Windows:**

     `<install_location>\Arcot Systems\sdk\java\properties\`
   - **Unix-Based Platforms:**

     `<install_location>/arcot/sdk/java/properties/`
5. Depending on the SDK you are configuring, open the required properties file:
   - `riskfort.risk-evaluation.properties`
   - `riskfort.issuance.properties`
6. Specify the following parameters in the file:
   - `TRANSPORT_TYPE`
   - `CA_CERT_FILE`
7. Save the changes and close the file.
8. Restart the RiskFort Server.

9. Restart the application server.

Book: Refer to Appendix F, "Configuring SSL" in the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for detailed information on how to configure SSL between different components of RiskFort.

Appendix B
Sample Code

This appendix provides the sample codes that you can run to test the following RiskFort functionality:

- User operations (Sample Code for User Operations)
- Risk evaluation and post-evaluation (Sample Code for Risk Evaluation and Post-Evaluation)

Sample Code for User Operations

You can use the following code sample to test the user-related operations (creation, reading, and update) in RiskFort database.

```
/* Packages to be imported for RiskFort Issuance API */

import java.util.Map;
import java.util.HashMap;

import com.arcot.riskfortissuanceAPI.User;
import com.arcot.riskfortissuanceAPI.UserManager;
import com.arcot.riskfortissuanceAPI.UserRepManager;
import com.arcot.riskfortissuanceAPI.UserRepResult;
import com.arcot.riskfortissuanceAPI.UserRepositoryException;
import com.arcot.riskfortissuanceAPI.UserRepositoryService;
import com.arcot.riskfortissuanceAPI.AdditionalInputs;

public class RiskFort_Issuance {

  // In this example values are hard coded for sample use.
  public static void main(String[] args) {
```

```
    /*
    initialize:
      Initializes API object from the input property file.
      initialize() should be called only once at the application startup.

      Following are the fields and format of the property file.
        HOST.1=RiskFort server IP address.
        PORT.1=RiskFort server port number.
        TRANSPORT_TYPE = Connection Type. Possible values are TLS/TCP.
        CA_CERT_FILE = Required if TRANSPORT_TYPE = TLS: CA certificate file.
server CA certificate (in PEM format) file path.

    public static synchronized void initialize(java.lang.String
propertyLocation)
            throws UserRepositoryException

    Parameters:
      propertyLocation - Represents the location to be passed as a parameter
with respect to the class path.
     If <code>null</code> is passed, it will take the default location which is
properties/riskfort_issuance.properties.
    Returns:

    Throws:
      UserRepositoryException
    */

    UserRepositoryService urs;

    // Sample code to initialize the API object from the input property file.
    String propertiesFileLocation= "/properties/riskfort_issuance.properties";

    try {
```

```
      System.out.println("Initializing RiskFort Issuance API using " +
propertiesFileLocation);


      // Initialize the API object from the input property file.
      UserRepManager.initialize(propertiesFileLocation);


      // Gets the implememtation of User Repository Service.
      urs = UserRepManager.getGrpUserRepService();


      System.out.println("RiskFort Issuance API initialized.");
     catch (UserRepositoryException e) {


      /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
     * String code = e.getCode();
     * String message = e.getMessage();
     */
      System.out.println("Exception during initialize.");
      System.out.println("Error code: " + e.getCode());
      System.out.println("Error message: " + e.getMessage());


      /* The following error codes are returned by the API. */


      /* ERROR_CONF_API
      * Possible Reason:
      *   The parameters in the configuration file are missing or not set
correctly.
      * Possible Action:
      *   All the parameters in the configuration file need to be set correctly.
      */
    }
```

```
    /*
    create:
      Creates a user in the system.
    public UserRepResult create(User user, AdditionalInputs additionalInputs)
          throws UserRepositoryException;

    Parameters:
      user - Represents a user object which is to be created in the system.
      additionalInputs - Additional inputs that may be needed for different
operations. This has been kept for future use.
    Returns:
      UserRepResult - It contains the response code and transaction identifier
generated at server.
    Throws:
      UserRepositoryException
    */

    // Sample code to create a user in the system.
    String userName = "TestUser";
    String orgName = "DEFAULTORG";
    String firstName = "Test";
    String lastName = "User";
    String email = "TestUser@abcd.com";
    String pam = "My PAM";

    Map additionalInputs = new HashMap();

    String extName1 = "AR_RF_CALLER_ID";
    String extValue1 = "test_caller";
    if(extName1 != null && extName1 != "" )
      additionalInputs.put(extName1, extValue1);
```

```
    // Create and initialize an object to hold the input user information to be
sent to the RiskFort Server.
    User user = null;


    // Create and initialize an object to hold the response from the RiskFort
Server.
    UserRepResult userRepRes = null;


    System.out.println("The following credentials are used to create a user.");
    System.out.println("Username: " + userName);
    System.out.println("Organization name: " + orgName);
    System.out.println("First name: " + firstName);
    System.out.println("Last name: " + lastName);
    System.out.println("Email: " + email);
    System.out.println("PAM: " + pam);


    try {
      // Gets the implementation user repository service.
      urs = UserRepManager.getGrpUserRepService();

      // Get the implementation of User interface.
      user = UserManager.getNewUserObject(userName, orgName);

      // Set the user properties
      user.setProperty(User.USER_PROPERTY_FIRSTNAME, firstName);
      user.setProperty(User.USER_PROPERTY_LASTNAME, lastName);
      user.setProperty(User.USER_PROPERTY_EMAILADDR, email);
      user.setPAM(pam);


      System.out.println("Calling create.");
      // Call the API to create the user and set the response of the RiskFort
Server in the userRepRes object.
      userRepRes = urs.create(user, additionalInputs);
```

```
      System.out.println("create succeeded.");

    } catch (UserRepositoryException e) {

      /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
      * String code = e.getCode();
      * String message = e.getMessage();
      */
      System.out.println("Exception in 'create'");
      System.out.println("Error code: " + e.getCode());
      System.out.println("Error message: " + e.getMessage());

      /* The following error codes are returned by the API. */

      /* INVALID_EMAIL
      * Possible Reason:
      *   The email set in the user object to the API is invalid.
      * Possible Action:
      *   Take suitable action to correct the email id.
      *   For example: Ask the end user to give a valid email-id and then pass
it to the API.
      */

      /* INVALID_ORG_NAME
      * Possible Reason:
      *   The organization name set in the user object to the API is invalid.
      * Possible Action:
      *   Set a valid organization name in the user object and call the API
again.
      *
      */
```

```
      /* INVALID_USER_OBJECT
      * Possible Reason:
      *   The user object passed as input to the API is invalid.
      * Possible Action:
      *   Check if the structure of the User object passed to the API is valid.
      *   Correct wherever necessary and call the API again.
      */

      /* MANDATORY_ATTRIB_EXCEEDS_SIZE
      * Possible Reason:
      *   One of the mandatory input attributes in the call to the API has
exceeded its size.
      * Possible Action:
      *   Take suitable action to limit the size of the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute has exceeded its size.
      */

      /* MANDATORY_ATTRIB_INVALID_CHAR
      * Possible Reason:
      *   One of the mandatory input attribute in the call to the API contains
invalid character(s).
      * Possible Action:
      *   Take suitable action to validate the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute contains the invalid character(s).
      */

      /* MANDATORY_ATTRIB_MISSING
      * Possible Reason:
      *   One of the mandatory input attributes in the call to the API is
missing.
      * Possible Action:
```

```
      *   Set the input missing attribute correctly.
      *   The getMessage() method of class UserRepositoryException tells which
mandatory attribute is missing.
      */


      /* OPTIONAL_ATTRIB_EXCEEDS_SIZE
      * Possible Reason:
      *   One of the optional input attributes in the call to the API has
exceeded its size.
      * Possible Action:
      *   Take suitable action to limit the size of the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute has exceeded its size.
      */


      /* OPTIONAL_ATTRIB_INVALID_CHAR
      * Possible Reason:
      *   One of the optional input attribute in the call to the API contains
invalid character(s).
      * Possible Action:
      *   Take suitable action to validate the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute contains the invalid character.
      */


      /* SERVER_INTERNAL_ERROR
      * Possible Reason:
      *   Server had a problem processing the transaction.
      * Possible Action:
      *   Check the Server logs for details.
      *   Report transaction failure and ask for a retry.
      */


      /* TCP_COMMUNICATION_ERROR
```

```
      * Possible Reason:
      *   Communication failure with RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* UNKNOWN_ERROR
      * Possible Reason:
      *   Corrupt response from the RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* USER_ALREADY_EXIST
      * Possible Reason:
      *   The user mentioned in the input to the API already exists.
      * Possible Action:
      *   Take suitable action.
      *   For example: Give this user a non-existing username or ask the
end-user for the same.
      */

      /* USERNAME_MISSING
      * Possible Reason:
      *   The username is not set in the input user object to the API.
      * Possible Action:
      *   Set the correct username attribute in the input user object of the API
      */
    }
    /*

    update:
      Updates a user in the system.
```

```
    public UserRepResult update(User user, AdditionalInputs additionalInputs)
      throws UserRepositoryException;


    Parameters:
     user - Represents a user object whose information has to be updated in the
system.
      additionalInputs - Additional inputs that may be needed for different
operations. This has been kept for future use.
    Returns:
      UserRepResult - It contains the response code and transaction identifier
generated at server.
    Throws:
      UserRepositoryException
    */


    // Sample code to update a user in the system.
    // Initialize an object to hold the input user information to be sent to the
RiskFort Server.
    user = null;
    // Initialize an object to hold the response from the RiskFort Server.
    userRepRes = null;


    // New pam to be updated for the specified user.
    pam = "New changed PAM";
    System.out.println("PAM changed to: " + pam);
    try {
      //Gets the implementation user repository service.
      urs = UserRepManager.getGrpUserRepService();


      // Get the implementation of User interface.
      user = UserManager.getUserDetails(userName, orgName);


      // Set the user properties to be updated.
      if ( firstName != null )
```

```
      user.setProperty(User.USER_PROPERTY_FIRSTNAME, firstName);
      if ( lastName != null )
      user.setProperty(User.USER_PROPERTY_LASTNAME, lastName);
      if ( email != null)
      user.setProperty(User.USER_PROPERTY_EMAILADDR, email);
      if ( pam != null)
      user.setPAM(pam);

      System.out.println("Calling update.");
      // Call the API to update the user and set the response of the RiskFort
Server in the userRepRes object.
      userRepRes = urs.update(user, additionalInputs);
      System.out.println("update succeeded.");

    } catch (UserRepositoryException e) {

      /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
      * String code = e.getCode();
      * String message = e.getMessage();
      */
      System.out.println("Exception in 'update'");
      System.out.println("Error code: " + e.getCode());
      System.out.println("Error message: " + e.getMessage());

      /* The following error codes are returned by the API. */

      /* INVALID_EMAIL
      * Possible Reason:
      *   The email set in the user object to the API is invalid.
      * Possible Action:
      *   Take suitable action to correct the email id.
```

```
      *   For example: Ask the end user to give a valid email-id and then pass
it to the API
      */

      /* INVALID_ORG_NAME
      * Possible Reason:
      *   The organization name set in the user object to the API is invalid.
      * Possible Action:
      *   Set a valid organization name in the user object and call the API
again.
      *
      */

      /* INVALID_USER_OBJECT
      * Possible Reason:
      *   The user object passed as input to the API is invalid.
      * Possible Action:
      *   Check if the structure of the User object passed to the API is valid.
      *   Correct wherever necessary and call the API again.
      */

      /* MANDATORY_ATTRIB_EXCEEDS_SIZE
      * Possible Reason:
      *   One of the mandatory input attributes in the call to the API has
exceeded its size.
      * Possible Action:
      *   Take suitable action to limit the size of the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute has exceeded its size.
      */

      /* MANDATORY_ATTRIB_INVALID_CHAR
      * Possible Reason:
```

```
      *   One of the mandatory input attribute in the call to the API contains
invalid character(s).

      * Possible Action:

      *   Take suitable action to validate the attribute.

      *   The getMessage() method of class UserRepositoryException tells which
attribute contains the invalid character(s).

      */


      /* MANDATORY_ATTRIB_MISSING

      * Possible Reason:

      *   One of the mandatory input attributes in the call to the API is
missing.

      * Possible Action:

      *   Set the input missing attribute correctly.

      *   The getMessage() method of class UserRepositoryException tells which
mandatory attribute is missing.

      */


      /* OPTIONAL_ATTRIB_EXCEEDS_SIZE

      * Possible Reason:

      *   One of the optional input attributes in the call to the API has
exceeded its size.

      * Possible Action:

      *   Take suitable action to limit the size of the attribute.

      *   The getMessage() method of class UserRepositoryException tells which
attribute has exceeded its size.

      */


      /* OPTIONAL_ATTRIB_INVALID_CHAR

      * Possible Reason:

      *   One of the optional input attribute in the call to the API contains
invalid character(s).

      * Possible Action:

      *   Take suitable action to validate the attribute.
```

```
      *   The getMessage() method of class UserRepositoryException tells which
attribute contains the invalid character(s).
      */

      /* SERVER_INTERNAL_ERROR
      * Possible Reason:
      *   Server had a problem processing the transaction.
      * Possible Action:
      *   Check the Server logs for details.
      *   Report transaction failure and ask for a retry.
      */

      /* TCP_COMMUNICATION_ERROR
      * Possible Reason:
      *   Communication failure with RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* UNKNOWN_ERROR
      * Possible Reason:
      *   Corrupt response from the RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* USER_NOT_FOUND
      * Possible Reason:
      *   The user mentioned in the input to the API does not exist.
      * Possible Action:
      *   Take suitable action.
      *   For example: check for the correctness of the username passed.
      */
```

```
      /* USERNAME_MISSING
      * Possible Reason:
      *   The username is not set in the input user object to the API.
      * Possible Action:
     *   Set the correct username attribute in the input user object of the API.
      */
    }

    /*
    read:
      Read user from the system. It will return the fully populated user object.
    public User read(java.lang.String userName, java.lang.String orgName,
AdditionalInputs additionalInputs)
        throws UserRepositoryException;

    Parameters:
      userName - Represents the username of the user whose details have to be
read from the system.
      orgName - Represents the organization to which the user belongs whose
details have to be read from the system.
      additionalInputs - Additional inputs that may be needed for different
operations. This has been kept for future use.
    Returns:
     User - Represents the user object whose information has been read from the
system.
    Throws:
      UserRepositoryException
    */


    // Sample code to read user details from the system.


    // initialize an object to hold the response from the RiskFort Server.
```

```
    user = null;


    try {
      //Gets the implementation user repository service.
      urs = UserRepManager.getGrpUserRepService();


      System.out.println("Calling read with username = " + userName + " and
organization name = " + orgName);


      // Call the API to read the user details from the system. This API will
return the fully populated user object.
      user  = urs.read(userName, orgName, additionalInputs);


      System.out.println("User details read are as follows: ");
      System.out.println("Username: " + userName);
      System.out.println("Organization name: " + orgName);
      System.out.println("First name: " + firstName);
      System.out.println("Last name: " + lastName);
      System.out.println("Email: " + email);
      System.out.println("PAM: " + pam);


    } catch (UserRepositoryException e) {


      /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
      * String code = e.getCode();
      * String message = e.getMessage();
      */
      System.out.println("Exception in 'read'");
      System.out.println("Error code: " + e.getCode());
      System.out.println("Error message: " + e.getMessage());


      /* The following error codes are returned by the API. */
```

```
      /* INVALID_ORG_NAME
      * Possible Reason:
      *   The organization name set in the user object to the API is invalid.
      * Possible Action:
      *   Set a valid organization name in the user object and call the API
again.
      */


      /* MANDATORY_ATTRIB_EXCEEDS_SIZE
      * Possible Reason:
      *   One of the mandatory input attributes in the call to the API has
exceeded its size.
      * Possible Action:
      *   Take suitable action to limit the size of the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute has exceeded its size.
      */


      /* MANDATORY_ATTRIB_INVALID_CHAR
      * Possible Reason:
      *   One of the mandatory input attribute in the call to the API contains
invalid character(s).
      * Possible Action:
      *   Take suitable action to validate the attribute.
      *   The getMessage() method of class UserRepositoryException tells which
attribute contains the invalid character(s).
      */


      /* MANDATORY_ATTRIB_MISSING
      * Possible Reason:
      *   One of the mandatory input attributes in the call to the API is
missing.
      * Possible Action:
```

```
      *   Set the input missing attribute correctly.
      *   The getMessage() method of class UserRepositoryException tells which
mandatory attribute is missing.
      */

      /* SERVER_INTERNAL_ERROR
      * Possible Reason:
      *   Server had a problem processing the transaction.
      * Possible Action:
      *   Check the Server logs for details.
      *   Report transaction failure and ask for a retry.
      */

      /* TCP_COMMUNICATION_ERROR
      * Possible Reason:
      *   Communication failure with RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* UNKNOWN_ERROR
      * Possible Reason:
      *   Corrupt response from the RiskFort Server.
      * Possible Action:
      *   Report transaction failure and ask for a retry.
      */

      /* USER_NOT_FOUND
      * Possible Reason:
      *   The user mentioned in the input to the API does not exist.
      * Possible Action:
      *   Take suitable action.
      *   For example: check for the correctness of the username passed.
```

```
      */


      /* USERNAME_MISSING
      * Possible Reason:
      *   The username is not set in the input user object to the API.
      * Possible Action:
     *   Set the correct username attribute in the input user object of the API.
      */
    }
  }
}
```

To Compile on Windows:

To compile this test program, save it in a file called `RiskFort_Issuance.java`. Make sure that the `arcot-riskfort-issuance.jar` and related jar files `arcot_core.jar`, `bcprov-jdk14-131.jar`, `commons-beanutils.jar`, `commons-collections-3.1.jar`, `commons-digester.jar`, `commons-lang-2.0.jar`, `commons-logging.jar`, `commons-pool.jar`, `dom4j-1.6.1.jar`, `json-lib-0.7.1.jar`, `log4j-1.2.9.jar`, and `servlet.jar` are in the JAVA compiler's `CLASSPATH`. Then run the JAVA compiler.

The `arcot-riskfort-issuance.jar` file is usually present in the `sdk\java\lib\`**`arcot`** directory in the RiskFort installation directory. If your `RiskFort_Issuance.java` file is saved in `\Program Files\ARCOT SYSTEMS\sdk\`**`java`**, use the following command shown below. If your JAVA file is not in this directory, provide the full `pathname` to the `sdk\java\`**`lib`** directory in `CLASSPATH`.

```
> cd \program files\arcot systems\sdk\java


> javac -classpath
".;lib\arcot\arcot-riskfort-issuance.jar;lib\arcot\arcot_core.jar;lib\ex
ternal\bcprov-jdk14-131.jar;lib\external\commons-beanutils.jar;lib\exter
nal\commons-collections-3.1.jar;lib\external\commons-digester.jar;lib\ex
ternal\commons-lang-2.0.jar;lib\external\commons-logging.jar;lib\externa
l\commons-pool.jar;lib\external\dom4j-1.6.1.jar;lib\external\json-lib-0.
7.1.jar;lib\external\log4j-1.2.9.jar;lib\external\servlet.jar;%CLASSPATH
%" RiskFort_Issuance.java
```

This creates the output file `RiskFort_Issuance.class` in the same directory as the JAVA file.

To Compile on UNIX Platforms

To compile this test program, save it in a file called RiskFort_Issuance.java. Make sure that the arcot-riskfort-issuance.jar and related jar files arcot_core.jar, bcprov-jdk14-131.jar, commons-beanutils.jar, commons-collections-3.1.jar, commons-digester.jar, commons-lang-2.0.jar, commons-logging.jar, commons-pool.jar, dom4j-1.6.1.jar, json-lib-0.7.1.jar, log4j-1.2.9.jar, and servlet.jar are in the JAVA compiler's CLASSPATH. Then run the JAVA compiler.

The arcot-riskfort-issuance.jar file is usually present in the sdk/java/lib/**arcot** directory in the RiskFort installation directory. If your RiskFort_Issuance.java file is saved in /opt/arcot/sdk/**java**, use the following command shown below. If your JAVA file is not in this directory, provide the full pathname to the sdk/java/**lib** directory in CLASSPATH.

```
> cd /opt/arcot/sdk/java
```

```
> javac -classpath
".:./lib/arcot/arcot-riskfort-issuance.jar:./lib/arcot/arcot_core.jar:./
lib/external/bcprov-jdk14-131.jar:./lib/external/commons-beanutils.jar:.
/lib/external/commons-collections-3.1.jar:./lib/external/commons-digeste
r.jar:./lib/external/commons-lang-2.0.jar:./lib/external/commons-logging
.jar:./lib/external/commons-pool.jar:./lib/external/dom4j-1.6.1.jar:./li
b/external/json-lib-0.7.1.jar:./lib/external/log4j-1.2.9.jar:./lib/exter
nal/servlet.jar:$CLASSPATH" RiskFort_Issuance.java
```

This creates the output file RiskFort_Issuance.class in the same directory as the JAVA file.

To Run on Windows

Before you can run the test, the Arcot RiskFort Service must be installed and started. To run the test, make sure that the SDK library is in the path and arcot-riskfort-issuance.jar is in the CLASSPATH then run the JAVA command as shown below.

To run the test, use the following commands:

```
> cd \program files\arcot systems\sdk\java
```

```
> java -classpath
".;lib\arcot\arcot-riskfort-issuance.jar;lib\arcot\arcot_core.jar;lib\ex
ternal\bcprov-jdk14-131.jar;lib\external\commons-beanutils.jar;lib\exter
nal\commons-collections-3.1.jar;lib\external\commons-digester.jar;lib\ex
ternal\commons-lang-2.0.jar;lib\external\commons-logging.jar;lib\externa
l\commons-pool.jar;lib\external\dom4j-1.6.1.jar;lib\external\json-lib-0.
7.1.jar;lib\external\log4j-1.2.9.jar;lib\external\servlet.jar;%CLASSPATH
%" RiskFort_Issuance
```

To Run on UNIX Platforms

Before you can run the test, the Arcot RiskFort Service must be installed and started. To run the test, make sure that the SDK library is in the path and `arcot-riskfort-issuance.jar` is in the `CLASSPATH` then run the JAVA command as shown below.

To run the test, use the following commands:

```
> cd /opt/arcot/sdk/java


> java -classpath
".:./lib/arcot/arcot-riskfort-issuance.jar:./lib/arcot/arcot_core.jar:./
lib/external/bcprov-jdk14-131.jar:./lib/external/commons-beanutils.jar:.
/lib/external/commons-collections-3.1.jar:./lib/external/commons-digeste
r.jar:./lib/external/commons-lang-2.0.jar:./lib/external/commons-logging
.jar:./lib/external/commons-pool.jar:./lib/external/dom4j-1.6.1.jar:./li
b/external/json-lib-0.7.1.jar:./lib/external/log4j-1.2.9.jar:./lib/exter
nal/servlet.jar:$CLASSPATH" RiskFort_Issuance
```

Expected Output

```
Initializing RiskFort Issuance API using
/properties/riskfort.issuance.properties
RiskFort Issuance API initialized.
The following credentials are used to create a user.
Username: TestUser
organization name = DEFAULTORG
First name: Test
Last name: User
Email: TestUser@abcd.com
PAM: My PAM
Calling create.
create succeeded.
PAM changed to: New changed PAM
Calling update.
update succeeded.
Calling read with username = TestUser and organization name = DEFAULTORG
User details read are as follows:
Username: TestUser
organization name = DEFAULTORG
First name: Test
Last name: User
Email: TestUser@abcd.com
PAM: New changed PAM
```

Sample Code for Risk Evaluation and Post-Evaluation

You can plug the following sample code snippet in to your application code to test the risk evaluation and post-evaluation functionality of RiskFort.

```
/* Packages to be imported for RiskFort Transaction API */
```

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Map;
import java.util.HashMap;

import com.arcot.riskfortAPI.DeviceContext;
import com.arcot.riskfortAPI.LocationContext;
import com.arcot.riskfortAPI.PostEvaluateResponse;
import com.arcot.riskfortAPI.RiskAssessment;
import com.arcot.riskfortAPI.RiskException;
import com.arcot.riskfortAPI.RiskFactory;
import com.arcot.riskfortAPI.RiskXActionAPI;
import com.arcot.riskfortAPI.TransactionContext;
import com.arcot.riskfortAPI.UserContext;
import com.arcot.riskfortAPI.AdditionalInputs;

public class Assess_Risk {

// In this example values are hard coded for sample use.
public static void main(String[] args) {
  /*
  initialize:
    Initializes API object from the input property file.
    initialize() should be called only once at the application startup.

    Following are the fields and format of the property file.
      HOST.1=RiskFort server IP address
      PORT.1=RiskFort server port number
      TRANSPORT_TYPE = Connection Type. Possible values are TLS/TCP
      CA_CERT_FILE = Required if TRANSPORT_TYPE = TLS: CA certificate file.
server CA certificate (in PEM format) file path.
```

```
  public static synchronized void initialize(String propertyLocation)
    throws IOException, RiskException;


  Parameters:
      propertyLocation - Represents the location to be passed as a parameter
with respect to the class path.
      If <code>null</code> is passed, it will take default location is
properties/riskfort.risk-evaluation.properties.
  Returns:


  Throws:
    RiskException - If request fails for any reason.
    IOException
  */



  // Sample code to initialize the API object from the input property file.
  // Create a RiskXActionAPI object.
  RiskXActionAPI api = null;


  String propertiesFileLocation =
"/properties/riskfort.risk-evaluation.properties";


  try {
      System.out.println("Initializing RiskFort API using " +
propertiesFileLocation);


    // Initializes RiskXActionAPI object from the input property file.
      RiskFactory.initialize(propertiesFileLocation);
      //  Get RiskXActionAPI object that has been initialized earlier.
      api = RiskFactory.getRiskXActionAPI();


      System.out.println("RiskFort API initialized.");
  catch (IOException e) {
```

```
    // Take suitable action.


  } catch (RiskException e) {


    /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
    * String code = e.getErrorCode();
    * String message = e.getMessage();
    */
    System.out.println("Exception during initialize.");
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println("Error message: " + e.getMessage());


    /* The following error codes are returned by the API. */


    /* ERRCODE_INVALID_INPUT
    * This can be caused due to two reasons.
    * 1. Possible Reason:
    *    The transport type mentioned in the properties file is invalid.
    *    Possible Action:
    *     Mention the valid transport type in the
"riskfort.risk-evaluation.properties" file.
    * 2. Possible Reason:
    *    If TRANSPORT_TYPE=TLS in the properties file, the primary key file used
for TLS was not found.
    *    Possible Action:
    *     Check for the correctness of the path to the primary key file.
    */


    /* ERRCODE_RISKXACTIONAPI_ALREADY_INITIALIZED
    * Possible Reason:
    *    The API being initialized has already been initialized.
    * Possible Action:
```

```
    *   Get the RiskXActionAPI object and continue with the transaction.
    */

    /* ERROR_CONF_INVALID_POOL
    * Possible Reason:
    *  Inability to create a pool of live connections to the RiskFort Server.
    * Possible Action:
    *
    */

    /* ERROR_CONF_NOT_AVAILABLE
    * This can be caused due to two reasons.
    * 1. Possible Reason:
    *    The properties file could not be read.
    *    Possible Action:
    *     Check for the correctness of the path to the properties file.
    * 2. Possible Reason:
    *    The Root CA for the server certificate is invalid.
    *    Possible Action:
    *     Get a valid server certificate.
    */
  }


  /*
  evaluateRisk:
    Evaluates risk associated with the transaction and returns an advice
accordingly. It also provides a new DeviceId.
    Actions to be taken by the calling application are:
      1. The output DeviceId should be stored on the user machine in some form.
Most common way is to store it as a HTTP cookie. Nevertheless, storing it as a
HTTP cookie has the risk of it being deleted when user deletes all cookies on
the machine.
      2. Retrieve the DeviceId from user machine and set it using setDeviceID.
```

```
      3. If RiskAdvice is equal to INCREASEAUTH, perform second authentication
and pass the result of the second authentication to RiskFort using PostEvaluate.


  public RiskAssessment evaluateRisk(java.lang.String callerId,
        DeviceContext deviceContext,
        LocationContext locationContext,
        UserContext userContext,
        TransactionContext transactionContext,
        AdditionalInputs additionalInputs)
     throws RiskException


  Parameters:
    callerId - An identifier as decided by the application calling the API for
it's own tracking.
    deviceContext - Device contextual information.
    locationContext - Location contextual information(IP address).
    userContext - User contextual information.
    transactionContext - Transaction contextual information.
    additionalInputs - Additional inputs that may be needed for different
operations
  Returns:
    RiskAssessment - Contains RiskAdvice, a new DeviceId which should be placed
on the user machine, a RiskScore and other transaction related information.
  Throws:
    RiskException - If request fails for any reason.
  */


  // Sample code to evaluate risk associated with the transaction.


  RiskAssessment riskAssessment = null;
  System.out.println("The following information is used to assess the risk
associated with the transaction.");


  // Build the context to be used for risk evaluation
```

```
  String callerId = "MyApplicationTrackingId"; // string used by the calling
application for tracking across calls.


  // input user related information.
  UserContext userContext = new UserContext();


  //Unique identifier for the user. For example, in case of a bank it may be
user's bank account number.
  userContext.setUserId("TestUser");
  userContext.setOrg("DEFAULTORG");
  System.out.println("Username: " + userContext.getUserID());
  System.out.println("Organization Name:" + userContext.getOrg());


  // input device related information
  DeviceContext deviceContext = new DeviceContext();


  // JSON Signature comes from mfp_json.js.
  String jsonSignature =
"{\"navigator\":{\"platform\":\"Win32\",\"appName\":\"Netscape\",\"appCodeName
\":\"Mozilla\",\"appVersion\":\"5.0 (Windows;
en-US)\",\"language\":\"en-US\",\"oscpu\":\"Windows NT
5.0\",\"vendor\":\"\",\"vendorSub\":\"\",\"product\":\"Gecko\",\"productSub\":
\"20070312\",\"securityPolicy\":\"\",\"userAgent\":\"Mozilla/5.0 (Windows; U;
Windows NT 5.0; en-US; rv:1.8.0.11) Gecko/20070312
Firefox/1.5.0.11\",\"cookieEnabled\":true,\"onLine\":true},\"plugins\":[{\"nam
e\":\"Adobe Acrobat Plugin\",\"version\":\"7.00\"},{\"name\":\"Macromedia
Director\",\"version\":\"10.1\"},{\"name\":\"Windows Media Player Plug-in
Dynamic Link Library\",\"version\":\"\"},{\"name\":\"Macromedia Shockwave
Flash\",\"version\":\"9.0\"},{\"name\":\"Java Virtual
Machine\",\"version\":\"1.6.0\"}],\"screen\":{\"availHeight\":690,\"availWidth
\":1024,\"colorDepth\":32,\"height\":768,\"pixelDepth\":32,\"width\":1024},\"e
xtra\":{\"javascript_ver\":\"1.6\",\"timezone\":-330}}";


  deviceContext.buildDeviceSignature(jsonSignature, null, null);
  System.out.println("Device Signature: " +
deviceContext.getDeviceSignature());


  // Set the device id
```

```
  String idType = "HTTP_COOKIE";
  /* During the first call to evaluateRisk, deviceId=null as the device is not
recognized by the RiskFort server.
   * The RiskFort server then sets a deviceId in a cookie on the user's machine
which is passed to the RiskFort server during subsequent transactions.
  */
  String deviceId = null;
  deviceContext.setDeviceID(idType, deviceId);

  /* For each transaction, either the deviceId or the aggregatorID but not both
should to be set.

deviceContext.setAggregatorID("LcPywTghrtyed6KDuRcMbWiFFTYR2oFThfdDOtBKqKcdEXs
H9dFIFfrr/dsfdud");

    System.out.println("Aggregator ID: " + deviceContext.getAggregatorID());
  */

  // input location related information.
  LocationContext locationContext = new LocationContext();

  InetAddress ipAddress = null;
  // IP address of the user's machine, typically extracted from the HTTP header.
  try {
      ipAddress = InetAddress.getByName("10.150.10.150");
  } catch (UnknownHostException e) {
      // Take suitable action.
  }

  locationContext.setIpAddress(ipAddress);
  System.out.println("Ip address: " + locationContext.getIPAddress());

  // input transaction related information.
  TransactionContext transactionContext = new TransactionContext();
```

```
  transactionContext.setAction("action");
  transactionContext.setChannel("DEFAULT");

  /*For each transaction, either the extensible elements must be set in the
transaction context or the additional inputs must be set.

transactionContext.setExtensibleElements("MerchantID=id;MerchantCountry=countr
y;MerchantName=name");
  */

  Map additionalInputs = new HashMap();

  String extName1 = "MerchantID";
  String extValue1 = "id";
  String extName2 = "MerchantCountry";
  String extValue2 = "country";
  String extName3 = "MerchantName";
  String extValue3 = "name";
  if(extName1 != null && extName1 != "" )
    additionalInputs.put(extName1, extValue1);
  if(extName2 != null && extName2 != "" )
    additionalInputs.put(extName2, extValue2);
  if(extName3 != null && extName3 != "" )
    additionalInputs.put(extName3, extValue3);

  try {
    System.out.println("evaluateRisk called.");
    // Call the API to evaluate the risk associated with the transaction.
    riskAssessment  = api.evaluateRisk(callerId, deviceContext,
locationContext, userContext, transactionContext, additionalInputs);

    System.out.println("evaluateRisk succeeded.");
```

```
    System.out.println("Device Id set on the user's machine: " +
riskAssessment.getOutputDeviceId());
  } catch (RiskException e) {
    /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
    * String code = e.getErrorCode();
    * String message = e.getMessage();
    */
    System.out.println("Exception in 'evaluateRisk'.");
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println("Error message: " + e.getMessage());

    /* The following error codes are returned by the API. */

    /* ERRCODE_INVALID_PACKET_FROM_SERVER
    * Possible Reason:
    *   Invalid Packet type received from the server.
    * Possible Action:
    *   Report transaction failure and ask for a retry.
    */

    /* ERRCODE_PARSING_DATA
    * Possible Reason:
    *   Error in parsing the xml from the server.
    * Possible Action:
    *   Report transaction failure and ask for a retry.
    */
  }


  /*
  postEvaluate:
```

```
    Helps to make the final decision on the transaction based on the output of
evaluateRisk and any second authentication that may have been performed by the
calling application.

    Also takes care of updating information in the RiskFort system as needed.


  public PostEvaluateResponse postEvaluate(java.lang.String callerId,

       RiskAssessment riskAssessment,

       boolean secondaryAuthenticationStatus,

       java.lang.String associationName,

       AdditionalInputs additionalInputs)

       throws RiskException;



  Parameters:

    callerId - An identifier as decided by the application calling the API for
it's own tracking.

    riskAssessment - The output from evaluateRisk.

    secondaryAuthenticationStatus - Result of second authentication.

        Pass "true" if secondary authentication succeeded, "false" otherwise.

        If evaluateRisk returned an advice other than INCREASEAUTH (i.e.
secondary authentication was not asked for), pass "false".


    associationName - A value that user chose as the association name for the
machine from where the transaction has been carried out.

        User should be recommended not to choose association for shared
machines, in which case "null" can be passed.

    additionalInputs - Additional inputs that may be needed for different
operations. This has been kept for future use.

  Returns:

    PostEvaluateResponse - Indicates whether or not this transaction should be
allowed to continue. Can be checked using isAllowAdvised() which returns "true"
if the transaction should be allowed and "false" if it should be denied.


  Throws:

    RiskException - If request fails for any reason.
```

```
  */

  // Sample code to make the final decision on the transaction based on the
output of evaluateRisk and any second authentication that may have been
performed by the calling application.
  // Here the RiskAssessment object passed as input is the object returned by
the call to evaluateRisk()

  PostEvaluateResponse postEvalResponse = null;

 String associationName; // the association name for the machine from where the
transaction has been carried out

 boolean secondaryAuthenticationStatus; // Result of any second authentication
that may have been performed by the calling application.

  // Build the context to be used in the postEvaluate call.
  associationName = "testAssociationName";
  secondaryAuthenticationStatus = true;
  try {
    System.out.println("Calling postEvaluate with Secondary Authentication
Status = " + secondaryAuthenticationStatus );
    System.out.println("Association name passed: " + associationName);

    // Call the API to make the final decision based on evaluateRisk and second
Authentication.
    postEvalResponse = api.postEvaluate(callerId, riskAssessment,
secondaryAuthenticationStatus, associationName, additionalInputs);

    System.out.println("postEvaluate succeeded.");
  } catch (RiskException e) {

    /* The following methods on UserRepositoryException object can be used to
get the error codes and error messages as follows:
    * String code = e.getErrorCode();
```

```
    * String message = e.getMessage();
    */
    System.out.println("Exception in 'postEvaluate'.");
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println("Error message: " + e.getMessage());

    /* The following error codes are returned by the API. */

    /* ERRCODE_INVALID_PACKET_FROM_SERVER
    * Possible Reason:
    *    Invalid Packet type received from the server.
    * Possible Action:
    *    Report transaction failure and ask for a retry.
    */

    /* ERRCODE_PARSING_DATA
    * Possible Reason:
    *    Error in parsing the xml from the server.
    * Possible Action:
    *    Report transaction failure and ask for a retry.
    */
  }
  System.out.println("Risk Evaluation done.");
  }
}
```

To Compile on Windows:

To compile this test program, save it in a file called `Assess_Risk.java`. Make sure that the `arcot-riskfort-issuance.jar` and related jar files `arcot_core.jar`, `arcot-riskfort-mfp.jar`, `bcprov-jdk14-131.jar`, `commons-beanutils.jar`, `commons-collections-3.1.jar`, `commons-digester.jar`, `commons-lang-2.0.jar`,

commons-logging.jar, commons-pool.jar, dom4j-1.6.1.jar, json-lib-0.7.1.jar, log4j-1.2.9.jar, and servlet.jar are in the JAVA compiler's CLASSPATH. Then run the JAVA compiler.

The arcot-riskfort-issuance.jar file is usually present in the sdk\java\lib**arcot** directory in the RiskFort installation directory. If your Assess_Risk.java file is saved in \Program Files\Arcot Systems\sdk**java**, use the following command shown below. If your JAVA file is not in this directory, provide the full pathname to the sdk\java**lib** directory in CLASSPATH.

```
> cd \program files\arcot systems\sdk\java
```

```
> javac -classpath
".;lib\arcot\arcot-riskfort-issuance.jar;lib\arcot\arcot_core.jar;lib\ar
cot\arcot-riskfort-mfp.jar;lib\external\bcprov-jdk14-131.jar;lib\externa
l\commons-beanutils.jar;lib\external\commons-collections-3.1.jar;lib\ext
ernal\commons-digester.jar;lib\external\commons-lang-2.0.jar;lib\externa
l\commons-logging.jar;lib\external\commons-pool.jar;lib\external\dom4j-1
.6.1.jar;lib\external\json-lib-0.7.1.jar;lib\external\log4j-1.2.9.jar;li
b\external\servlet.jar;%CLASSPATH%" Assess_Risk.java
```

This creates the output file Assess_Risk.class in the same directory as the JAVA file.

To Compile on UNIX Platforms

To compile this test program, save it in a file called Assess_Risk.java. Make sure that the arcot-riskfort-issuance.jar and related jar files arcot_core.jar, arcot-riskfort-mfp.jar, bcprov-jdk14-131.jar, commons-beanutils.jar, commons-collections-3.1.jar, commons-digester.jar, commons-lang-2.0.jar, commons-logging.jar, commons-pool.jar, dom4j-1.6.1.jar, json-lib-0.7.1.jar, log4j-1.2.9.jar, and servlet.jar are in the JAVA compiler's CLASSPATH. Then run the JAVA compiler.

The arcot-riskfort-issuance.jar file is usually present in the sdk/java/lib/**arcot** directory in the RiskFort installation directory. If your Assess_Risk.java file is saved in /opt/arcot/sdk/**java**, use the following command shown below. If your JAVA file is not in this directory, provide the full pathname to the sdk/java/**lib** directory in CLASSPATH.

```
> cd /opt/arcot/sdk/java
```

```
> javac -classpath
".:./lib/arcot/arcot-riskfort-issuance.jar:./lib/arcot/arcot_core.jar:./
lib/arcot/arcot-riskfort-mfp.jar:./lib/external/bcprov-jdk14-131.jar:./l
ib/external/commons-beanutils.jar:./lib/external/commons-collections-3.1
.jar:./lib/external/commons-digester.jar:./lib/external/commons-lang-2.0
```

```
.jar:./lib/external/commons-logging.jar:./lib/external/commons-pool.jar:
./lib/external/dom4j-1.6.1.jar:./lib/external/json-lib-0.7.1.jar:./lib/e
xternal/log4j-1.2.9.jar:./lib/external/servlet.jar:$CLASSPATH"
Assess_Risk.java
```

This creates output file `Assess_Risk.class` in the same directory as the JAVA file.

To Run on Windows

Before you can run the test, the Arcot RiskFort Service must be installed and started. To run the test, make `sure` that the SDK library is in the path and `arcot-riskfort-issuance.jar` is in the `CLASSPATH` then run the JAVA command as shown below.

To run the test, use the following commands:

```
> cd \program files\arcot systems\sdk\java


> java -classpath
".;lib\arcot\arcot-riskfort-issuance.jar;lib\arcot\arcot_core.jar;lib\ar
cot\arcot-riskfort-mfp.jar;lib\external\bcprov-jdk14-131.jar;lib\externa
l\commons-beanutils.jar;lib\external\commons-collections-3.1.jar;lib\ext
ernal\commons-digester.jar;lib\external\commons-lang-2.0.jar;lib\externa
l\commons-logging.jar;lib\external\commons-pool.jar;lib\external\dom4j-1
.6.1.jar;lib\external\json-lib-0.7.1.jar;lib\external\log4j-1.2.9.jar;li
b\external\servlet.jar;%CLASSPATH%" Assess_Risk
```

To Run on UNIX Platforms

Before you can run the test, the Arcot RiskFort Service must be installed and started. To run the test, make sure that the SDK library is in the path and `arcot-riskfort-issuance.jar` is in the `CLASSPATH` then run the JAVA command as shown below.

To run the test, use the following commands:

```
> cd /opt/arcot/sdk/java


> java -classpath
".:./lib/arcot/arcot-riskfort-issuance.jar:./lib/arcot/arcot_core.jar:./
lib/arcot/arcot-riskfort-mfp.jar:./lib/external/bcprov-jdk14-131.jar:./l
ib/external/commons-beanutils.jar:./lib/external/commons-collections-3.1
.jar:./lib/external/commons-digester.jar:./lib/external/commons-lang-2.0
.jar:./lib/external/commons-logging.jar:./lib/external/commons-pool.jar:
./lib/external/dom4j-1.6.1.jar:./lib/external/json-lib-0.7.1.jar:./lib/e
xternal/log4j-1.2.9.jar:./lib/external/servlet.jar:$CLASSPATH"
Assess_Risk
```

Expected Output

```
Initializing RiskFort API using /properties/riskfort.risk-evaluation.properties

RiskFort API initialized.

The following information is used to assess the risk associated with the
transaction.

Username: TestUser

organization name = DEFAULTORG

Device Signature:
{"DEVICESIG":{"VERSION":"1.0","OS_BROWSER":{"browser_ver":"1.5.0.11","os":"Win
dows NT
5.0","browser":"Netscape","javascript_ver":"1.6"},"SCREEN":{"colorDepth":"32",
"availWidth":"1024","width":"1024","height":"768","availHeight":"690"},"HTTP_H
EADER":{"user-agent":"Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US;
rv:1.8.0.11) Gecko/20070312
Firefox/1.5.0.11"},"OPTIONAL":{"SOFTWARE":[{"name":"Adobe Acrobat
Plugin","version":"7.00"},{"name":"Macromedia
Director","version":"10.1"},{"name":"Windows Media Player Plug-in Dynamic Link
Library"},{"name":"Macromedia Shockwave Flash","version":"9.0"},{"name":"Java
Virtual
Machine","version":"1.6.0"}]},"USER_PREF":{"sys_lang":"en-us","timezone":"-330
","user_lang":"en-us"}}}

Ip address: /10.150.10.150

evaluateRisk called.

evaluateRisk succeeded.

Device Id set on the user's machine:
EGPbuaASigkCN0qLiZePTTbpD4SZIQWS5CNlbEHVhrDVR1nd0lsni9cc/vpvtppg

Calling postEvaluate with Secondary Authentication Status = true

Association name passed: testAssociationName

postEvaluate succeeded.

Risk Evaluation done.
```

Appendix C
Java API Reference

The RiskFort SDK constitutes a set of Java classes and methods that provide a way for your online application to programmatically integrate with RiskFort objects. The RiskFort Java SDK consists of the following components:

- The Risk Evaluation Java classes
- The Issuance Java classes
- Javadoc information for the associated Java classes and methods

Note: The Sample Application shipped with RiskFort demonstrates the usage of the Java classes and methods.See Appendix G, "RiskFort Sample Application" for more information on RiskFort Sample Application.

Accessing the Javadoc HTML Documentation

You can use the Javadoc information provided with the RiskFort SDK along with this guide and other Java reference materials, to add RiskFort risk evaluation and issuance services to new or existing Java applications.

If you are updating an existing RiskFort application, then you must consult the Release Notes and Javadoc HTML documentation for deprecated Java APIs before making changes.

You can copy the Javadoc HTML information, consisting of the entire directory tree, to another location on your development system and then uninstall the RiskFort SDK and runtime components. Alternatively, you can also access the Javadocs directly from the RiskFort package.

See Table C-1 on page C-118 and Table C-3 on page C-122 for the Javadoc installation locations.

Risk Evaluation API

Table C-1 lists the files that are installed as a part of the Risk Evaluation SDK component. The base location for these files is:

Windows

`<install_location>\Arcot Systems\`

UNIX Platforms

`<install_location>/arcot/`

Table C-1. Risk Evaluation API Files

Location	File Name	Description

`docs\riskfort\` (Windows)
 	
`docs/riskfort/` (UNIX Platforms)
	`Arcot-RiskFort-2.2.6-risk-evaluation-sdk-javadocs.zip`
	
	or
	
	`Arcot-RiskFort-2.2.6-risk-evaluation-sdk-javadocs.tar.gz`
		Javadoc HTML documentation for the Java classes and methods provided with the Risk Evaluation Java API.

`samples\java\` (Windows) `samples/java/` (UNIX Platforms)	`riskfort-2.2.6-sample-application.war`	The demonstration that illustrates the use of both Risk Evaluation and Issuance APIs. See Appendix G, "RiskFort Sample Application" for more information on how to run and use the Sample Application.
	`riskfort-2.2.6-sample-callouts.war`	The demonstration that illustrates the use of the Callout feature. **Book:** See Appendix A, "Working with Sample Callouts" in *Arcot RiskFort 2.2.6 Administration Guide* for more information on how to run and use the Sample Callouts.

Table C-1. Risk Evaluation API Files

Location	File Name	Description
sdk\java\lib**arcot**\ (Windows) sdk/java/lib/**arcot**/ (UNIX Platforms)	arcot_core.jar	The proprietary Java Archive (JAR) file containing the set of shared components, toolkits, and services used to build the Arcot products.
	arcot-pool.jar	The Java Archive (JAR) file containing the classes and methods required for connection pooling between the RiskFort resourcepack and the User Data Service (UDS) server.
	arcot-riskfort-evaluaterisk.jar	The Java Archive (JAR) file containing the classes and methods associated with the Risk Evaluation API.
	arcot-riskfort-mfp.jar	The Java Archive (JAR) file containing the classes and methods associated with Device ID and Machine FingerPrint (MFP) collection.
sdk\java**properties**\ (Windows) sdk/java/**properties**/ (UNIX Platforms)	log4j.properties.risk-evaluation	The properties file used by classes and methods associated with the Risk Evaluation API to specify the logging behavior.
	riskfort.risk-evaluation.properties	The properties file used by classes and methods associated with the Risk Evaluation API to read RiskFort Server information.

Third-Party Software Used by Risk Evaluation API

The Risk Evaluation API also uses the **third-party software** listed in Table C-2.

- **Windows**

  <*install_location*>\Arcot Systems**sdk**\java\lib**external**\

- **UNIX Platforms:**

  <*install_location*>/arcot/**sdk**/java/lib/**external**/

Table C-2. Third-Party Files Used by the Risk Evaluation API

File Name	Description
bcprov-jdk14-139.jar	The Bouncy Castle APIs for supporting cryptographic operations. *http://www.bouncycastle.org/latest_releases.html*
commons-beanutils-1.7.0.jar	The Apache wrappers around Java java.lang.reflect and java.beans packages. *http://commons.apache.org/beanutils/*
commons-collections-3.1.jar	The Apache package for supporting new interfaces, implementations, and utilities of JDK classes. *http://commons.apache.org/collections/*
commons-httpclient-3.1.jar	The Apache package for supporting client-side authentication, HTTP state management, and HTTP connection management. *http://hc.apache.org/httpclient-3.x/index.html*
commons-lang-2.0.jar	The Apache Java utility packages that provide support for string manipulation methods, basic numerical methods, object reflection, creation and serialization, and system properties. *http://commons.apache.org/lang/*
commons-logging-1.0.4.jar	The Apache package for supporting different logging implementations. *http://commons.apache.org/logging/*
commons-pool-1.4.jar	The Apache package for supporting object-pooling implementations. *http://commons.apache.org/pool/*
dom4j-1.6.1.jar	The open-source XML framework for processing XML with integrated XPath and support for DOM, SAX, JAXP and Java 2 Collections. *http://www.dom4j.org/*
jaxen-1.1-beta-8.jar	The universal Java XPath engine that can be used with different object models, including DOM, XOM, dom4j, and JDOM. *http://jaxen.codehaus.org/*
jdom-1.0.jar	Java-based solution for accessing, manipulating, and outputting XML data from Java code. *http://www.jdom.org/*

Table C-2. Third-Party Files Used by the Risk Evaluation API

File Name	Description
`json-lib-0.7.1.jar`	The Java library for transforming beans, maps, collections, Java arrays, and XML to JSON and back again to beans. *http://www.json.org/java/*
`log4j-1.2.9.jar`	The Apache package for controlling run-time logging behavior of applications. *http://logging.apache.org/log4j/1.2/index.html*
`servlet-api-2.4.jar`	The Java package for defining the methods that all servlets must implement. *http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/package-summary.html*
`oro-2.0.8.jar`	Text-processing Java classes that provide support for Perl5-compatible regular expressions, AWK-like regular expressions, glob expressions, and utility classes for performing substitutions, splits, and filtering filenames. *http://jakarta.apache.org/oro/*
`xalan-2.7.0.jar`	XSLT processor for transforming XML documents into HTML, text, or other XML document types. *http://xml.apache.org/xalan-j/*
`xercesImpl-2.6.2.jar`	Framework for building XML parser components and configurations. *http://xerces.apache.org/xerces2-j/*
`xml-apis-1.0.b2.jar`	Framework to create Java Management Extensions (JMX) compatible Model MBeans. *http://commons.apache.org/modeler/*
`xmlParserAPIs-2.6.2.jar`	Framework for building XML parser components and configurations. *http://xml.apache.org/commons/*
`xom-1.1.jar`	XML object model for processing XML with Java. *http://www.xom.nu/*

Issuance API

Table C-3 lists the files that are installed as a part of the Issuance SDK component. The base location for these files is:

- **Windows**

  `<install_location>\Arcot Systems\`

- **UNIX Platforms**

  `<install_location>/arcot/`

Table C-3. Issuance API Files

Location	File Name	Description

`docs\`**`riskfort`**`\` (Windows)
 	
`docs/`**`riskfort`**`/` (UNIX Platforms)
	`Arcot-RiskFort-2.2.6-issuance-sdk-javadocs.zip`
	
	or
	`Arcot-RiskFort-2.2.6-issuance-sdk-javadocs.tar.gz`
		Javadoc HTML documentation for the Java classes and methods provided with the Issuance Java API.

`samples\`**`java`**`\` (Windows) `samples/`**`java`**`/` (UNIX Platforms)	`riskfort-2.2.6-sample-application.war`	The demonstration that illustrates the use of both Risk Evaluation and Issuance APIs. See Appendix G, "RiskFort Sample Application" for more information on how to run and use the Sample Application.
`sdk\java\lib\`**`arcot`**`\` (Windows) `sdk/java/lib/`**`arcot`**`/` (UNIX Platforms)	`arcot_core.jar`	The proprietary Java Archive (JAR) file containing the set of shared components, toolkits, and services used to build the Arcot products.
	`arcot-pool.jar`	The Java Archive (JAR) file containing the classes and methods required for connection pooling between the RiskFort resourcepack and the User Data Service (UDS) server.
	`arcot-riskfort-issuance.jar`	The Java Archive (JAR) file containing the classes and methods associated with the Issuance API.

Table C-3. Issuance API Files

Location	File Name	Description
`sdk\java\properties\` (Windows)	`log4j.properties.riskfort-issuance`	The properties file used by classes and methods associated with the Issuance API to specify the logging behavior.
`sdk/java/properties/` (UNIX Platforms)	`riskfort.issuance.properties`	The properties file used by classes and methods associated with the Issuance API to read RiskFort Server information.

Third-Party Software Used by Issuance API

The Issuance API also uses all the **third-party software** listed in Table C-2, *except the following*:

- `json-lib-0.7.1.jar`
- `servlet-api-2.4.jar`

Appendix D
Exceptions and Error Codes

This appendix lists all exceptions and error codes thrown by the RiskFort SDKs:

- SDK Exceptions
- Error Codes

SDK Exceptions

The SDK error codes can be categorized as:

- Risk Evaluation Exceptions
- Issuance Exceptions

Risk Evaluation Exceptions

The RiskFort Risk Evaluation APIs throw `RiskException`, if a related operation fails. The `getErrorCode()` function of the `RiskException` class returns the string value corresponding to the type of error. The possible values are listed in Table D-1.

Table D-1. Risk Evaluation Error Codes

Error Code	Value	Description
ERRCODE_INVALID_INPUT	ERRCODE_INVALID_INPUT	The input that RiskFort Server received is not valid. **Note:** This error is not thrown frequently.
ERRCODE_INVALID_PACKET_FROM_SERVER	ERROR_INVALID_PACKET_FROM_SERVER	The Server packet is either malformed or is corrupted. **Note:** This error is thrown frequently.
ERRCODE_MISSING_PARAMETER_IN_XML	ERRCODE_MISSING_PARAMETER_IN_XML	One or more XML parameters are either incomplete or are missing. **Note:** This error is thrown frequently.

Table D-1. Risk Evaluation Error Codes

Error Code	Value	Description
ERRCODE_PARSING_DATA	ERROR_PARSING_DATA	RiskFort Server could not parse the specified data.
ERRCODE_RISKXACTIONAPI_ALREADY_INITIALIZED	ERRCODE_RISKXACTIONAPI_ALREADY_INITIALIZED	The RiskXActionAPI interface has already been initialized.
ERRCODE_RISKXACTIONAPI_NOT_INITIALIZED	ERRCODE_RISKXACTIONAPI_NOT_INITIALIZED	The RiskXActionAPI interface has not yet been initialized.
ERRCODE_TRANSACTIONID_NULL	ERROR_TRASACTIONID_NULL	The TransactionID value is null.
ERROR_CONF_INVALID_POOL	0135211004	The connection to RiskFort Server could not be established.
ERROR_CONF_NOT_AVAILABLE	0135211005	The configuration keys of properties that are used for configuring the API could not be read correctly from the properties file.

Issuance Exceptions

The RiskFort user management APIs throw UserRepositoryException, if a related operation fails. The getCode() function of the UserRepositoryException class returns the string value corresponding to the type of error. The possible values are listed in Table D-2.

Table D-2. Issuance Error Codes

Error Code	Value	Description
ERROR_CONF_API	14	The configuration keys of properties that are used for configuring the API could not be read correctly from the properties file.
INVALID_EMAIL	12	The specified email is not valid.
INVALID_ORG_NAME	1	The specified organization name is not valid.
INVALID_USER_OBJECT	5	The specified User object is not valid.
MANDATORY_ATTRIB_EXCEEDS_SIZE	8	The specified required attribute exceeds the allowed maximum size.
MANDATORY_ATTRIB_INVALID_CHAR	9	The specified required attribute contains invalid characters.
MANDATORY_ATTRIB_MISSING	7	A required attribute is missing.

Table D-2. Issuance Error Codes

Error Code	Value	Description
OPTIONAL_ATTRIB_EXCEEDS_SIZE	10	The specified optional attribute exceeds the allowed maximum size.
OPTIONAL_ATTRIB_INVALID_CHAR	11	The specified optional attribute contains invalid characters.
SERVER_INTERNAL_ERROR	3	The error was caused due to an internal server error.
TCP_COMMUNICATION_ERROR	15	A TCP-based communication error occurred.
UNKNOWN_ERROR	13	An unknown error occurred.
USER_ALREADY_EXIST	2	The specified user already exists in the RiskFort database.
USER_NOT_FOUND	4	The specified user not found in the RiskFort database.
USERNAME_MISSING	6	The `username` of the `User` object is missing.

Error Codes

Table D-3 lists the response codes, reason codes, the possible cause for the failure, and solution wherever applicable.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
0	0	The operation was successful.	NA.
1000	2002	There was an internal error.	**Possible Cause:** Unexpected internal error.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	0	Value of one of the parameters used in the operation is invalid.	**Possible Cause:** The value of the parameter passed to the API is invalid. For example, the allowed values for user status are 0 and 1. If you set the value of this as 5, then you will get this error. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.
	2050	Value of one of the parameters used in the operation is empty.	**Possible Cause:** The parameter passed to the API is empty. **Solution:** Provide a non-empty value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	2051	Length of one of the parameters used in the operation has exceeded the maximum allowed value. **Note:** Length here refers to length of the parameter, for example password length.	**Possible Cause:** The length of the parameter passed to the API has exceeded the maximum value. **Solution:** Provide the parameter such that its length is less than or equal to the maximum allowed value. See Appendix E, "Input Data Validations" for the supported parameter values.
	2052	Length of one of the parameters used in the operation is less than minimum allowed value.	**Possible Cause:** The length of the parameter passed to the API is less than minimum value. **Solution:** Provide the parameter such that the length of the parameter is greater than or equal to the minimum allowed value. See Appendix E, "Input Data Validations" for the supported parameter values.
	2053	Value of one of the parameters used in the operation exceeded the maximum allowed value. **Note:** Value here refers to the value of the parameter.	**Possible Cause:** The value of the parameter passed to the API has exceeded the maximum allowed value. **Solution:** Provide the parameter such that the value of the parameter is less than or equal to the maximum allowed value. See Appendix E, "Input Data Validations" for the supported parameter values.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	2054	Value of one of the parameters used in the operation is less than the minimum allowed value.	**Possible Cause:** The value of the parameter passed to the API is less than the minimum allowed value. **Solution:** Provide the parameter such that the value of the parameter is greater than or equal to the minimum allowed value. See Appendix E, "Input Data Validations" for the supported parameter values.
	2055	Value of one of the parameters used in the operation is invalid.	**Possible Cause:** The value of the parameter passed to the API is invalid. For example, the allowed values for user status are 0 and 1. If you set the value of this as 5, then you will get this error. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.
	2056	Value of one of the parameters used in the operation contains invalid characters.	**Possible Cause:** The parameter specified by ParameterKey contains invalid characters. **Solution:** Provide valid characters for the parameter that is specified by ParameterKey.
	2057	One of the parameters used in the operation does not meet the formatting requirements.	**Possible Cause:** The parameter specified by ParameterKey has invalid format. **Solution:** Provide valid format for the parameter that is specified by ParameterKey.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	2061	Value of one of the parameters used in the operation is not allowed.	**Possible Cause:** The parameter specified by ParameterKey has invalid format. **Solution:** Provide valid format for the parameter that is specified by ParameterKey.
	8104	The specified Callout URL is not valid.	**Possible Cause:** The specified URL is incorrect. **Solution:** Provide the valid URL.
	8105	The specified duration is not valid.	**Possible Cause:** The Start Date is greater than the End Date. The specified value for Start Date and/or the End Date is in the past. **Solution:** The Start Date must be greater than the End Date and these should be the current or future dates (as this is the duration for the exception user).

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7501	0	The current operation on the database failed.	**Possible Cause:** Database is not running. **Solution:** Start the database. **Possible Cause:** Connection between the server and database is not complete. **Solution:** Establish the connection between server and database again. **Possible Cause:** The operation failed because of an internal error. **Solution:** Check the database logs for details and ensure appropriate action is taken based on these logs.
7502		An exception occurred because of an unexpected internal error.	**Possible Cause:** Internal error because of unexpected server behavior. **Solution:** Most likely cause might be server or database failure. Check the server transaction and database logs for details and ensure appropriate action is taken based on the server logs.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7503	0	Could not fetch time.	**Possible Cause:** The database settings are not set correctly in `arcotcommon.ini`. **Solution:** Verify and correct the database- related parameters in the file.
7601		Value of one of the parameters used in the operation does not exist.	**Possible Cause:** The value of the parameter passed to the API does not exist. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.
7602		The name specified for the new ruleset already exists.	**Possible Cause:** The value of the parameter passed to the API already exists. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.
7603		No Active ruleset of the specified name was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.
7604		No Proposed ruleset of the specified name was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7605	0	The name specified for the ruleset does not exist.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7606	8101	The *data* sharing type refers to another ruleset. Therefore, the addition operation is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data cannot be added to the other ruleset.
7607		The *data* sharing type refers to another ruleset. Therefore, the operation to set the value(s) is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data cannot be set to another ruleset.
7608		The *data* sharing type refers to another ruleset. Therefore, the delete operation is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data cannot be deleted from the other ruleset.
7609		The *data* sharing type refers to another ruleset. Therefore, the update operation is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data cannot be updated.
7610		The *data* sharing type refers to another ruleset. Therefore, the value rotation is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data values cannot be rotated.
7611		The *parameter* sharing type refers to another ruleset. Therefore, the operation to set the value(s) is not allowed.	**Possible Cause:** The rule or the ruleset refers to another ruleset. Therefore, the data cannot be set to another ruleset.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7612	0	No Active or Proposed data was found for the specified Negative Country.	**Possible Cause:** The value of the parameter passed to the API was not found in the ARRFNEGATIVECOUNTRYLIST table. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7613 7617		No Active or Proposed data was found for the specified Negative IP address.	**Possible Cause:** The value of the parameter passed to the API was not found in the ARRFUNTRUSTEDIPLIST table. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7614 7616		No Active or Proposed data was found for the specified Trusted IP address.	**Possible Cause:** The value of the parameter passed to the API was not found in the ARRFTRUSTEDIPLIST table. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7615		No Active or Proposed data was found for the specified rule.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7618	0	No Active or Proposed data was found for the specified Scoring configuration.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7619		No Active or Proposed data was found for the specified Execution configuration.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7620		The value of the configuration state parameter used in the operation is invalid.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7621		The ruleset cannot be created because the specified parameters and their values for the rule do not match.	**Possible Cause:** The parameters and their corresponding values passed to the API are invalid. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7622 7625	0	No Active ruleset found for the specified otherOrgName and otherConfigName parameters.	**Possible Cause:** The value of the otherOrgName and otherConfigName parameters passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7623 7626		No Active ruleset found for the specified data for otherOrgName and otherConfigName parameters.	**Possible Cause:** The value of the otherOrgName and otherConfigName parameters passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7624		The update is not allowed because it will create a cyclic dependency.	**Possible Cause:** The rule or the ruleset refers to another ruleset.
7627	8102	The specified country cannot be deleted because no corresponding Proposed data was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7628	8102	The specified IP range cannot be deleted because no corresponding Proposed data was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7629		The specified IP range cannot be deleted for the Trusted Aggregator because no corresponding Proposed data was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7630		The specified IP range cannot be deleted for the Trusted Aggregator because no corresponding Proposed data was found.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7631	0	The specified IP range does not exist in the Negative IP Address list.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7632	0	No Active or Proposed data for Trusted Aggregator was found for the specified Trusted IP address.	**Possible Cause:** The value of the parameter passed to the API was not found in the ARRFTRUSTEDIPLIST table. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7633	0	No Proposed data for the specified Trusted Aggregator was found.	**Possible Cause:** The value of the parameter passed to the API was not found in the ARRFTRUSTEDIPLIST table. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7634	0	The specified Trusted Aggregator does not exist in the Trusted IP address list.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7635	0	The specified Trusted Aggregator cannot be added, because it already exists in the Trusted IP address list.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7636	0	The specified Aggregator ID could not be generated because of an internal server error.	**Possible Cause:** Aggregator ID generation failed due to internal error in the server. **Solution:** Most likely cause might be because of database failure. Check the server transaction logs for details and ensure appropriate action is taken based on the server logs.
7637	0	The encryption key was not found in the database.	**Possible Cause:** The incursion key does not exist. **Solution:** Ensure that the key that you are using is correct.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7638	0	The supported port types could not be tokenized.	Possible Cause: Server host or port, or both might not be configured correctly. **Solution:** Provide correct host and port number. **Possible Cause:** Server might not be running. **Solution:** Start the server. **Possible Cause**: If SSL is configured, then certificates might not be configured correctly. **Solution:** Configure the TLS certificates correctly.
7639	0	The specified SSL Trust Store does not exist.	**Possible Cause:** The value of the parameter passed to the API was not found. **Solution:** Provide valid value for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values **Possible Cause:** The provided Trust Store name is not valid. **Solution:** You must provide a valid Trust Store name. **Possible Cause:** The provided organization name is not valid. **Solution:** You must provide a valid organization name.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7640	0	The specified score ranges were not found.	**Possible Cause:** No Score ranges or Advice were found in the ARRFADVICECONFIG and ARRFADVICECODE tables. The RiskFort database scripts were not run properly. **Solution:** Contact Arcot Professional Services (at ps@arcot.com) to resolve this issue.
7641	0	The Scoring Callout does not have the highest execution priority.	**Possible Cause:** The priority set in the Web service call is higher than the existing highest execution priority in the RiskFort database. **Solution:** Set the priority in the Web service call correctly.
7642	0	The Score does not have the highest execution priority.	**Possible Cause:** The priority set in the Web service call is higher than the existing highest execution priority in the RiskFort database. **Solution:** Set the priority in the Web service call correctly.
7643	8103	The required Add-On rule details were not specified. Therefore, the rule was not added.	**Possible Cause:** The Add-On rule details were not specified correctly in the web service call. **Solution:** Set the Web service call input parameters correctly.
7644	0	The specified Rule Type is not valid.	**Possible Cause:** The specified Rule Type is either not present in the ARRFADDONRULETYPE table or is an invalid rule type. **Solution:** You must provide a valid Rule Type.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7645	0	The Add-On rule was not added because it will create a cyclic dependency of *data*.	**Possible Cause:** The creation of the Add-On rule referring to another rule creates a cyclic dependency for the data that you added. Therefore, this is not allowed.
7646		The Add-On rule was not added because it will create a cyclic dependency of parameters.	**Possible Cause:** The creation of the Add-On rule referring to another rule creates a cyclic dependency for the parameters that you added. Therefore, this is not allowed.
7647		The rule that you specified is not available in the ruleset.	**Possible Cause:** The value of the rule passed to the API was not found in the specified ruleset. **Solution:** Provide valid rule for the parameter. See Appendix E, "Input Data Validations" for the supported parameter values
7648		Error occurred while fetching the parameters.	
7649		Other rules are dependant on this rule. Therefore, this rule cannot be deleted.	**Possible Cause:** Rules in another ruleset are currently referring to this rule. Therefore, you cannot delete this rule.
7650		The `TypeName` input is not specified in the call.	**Possible Cause:** The `TypeName` has not been specified in the Web service call. **Solution:** Specify the correct input value for `TypeName` in the input.
7651		Specified `TypeName` already exists.	**Possible Cause:** The `TypeName` in the input is already present in the system. **Solution:** Specify a different `TypeName` in the input.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7652	0	A cyclic redundancy is created with this update. Therefore, the update is not allowed. Some (as mentioned) rulesets have been migrated.	**Possible Cause:** A cyclic dependency will be created after the migration of *all* the required rulesets. As a result, only a few rulesets have been migrated. **Solution:** Change the ruleset(s) that have not yet been migrated, so that the cyclic dependency is eliminated.
7653		A cyclic redundancy is created with this update. Hence this update is not allowed. No rulesets have been migrated.	**Possible Cause:** A cyclic dependency will be created after the migration of *all* the required rulesets. **Solution:** Change the required ruleset(s) such that the cyclic dependency is eliminated.
7654		The Database operation failed while migrating to production. Some (as mentioned) rulesets have been migrated	**Possible Cause:** A cyclic dependency was found during migration of a ruleset. Therefore, only a few rulesets have been migrated. Some ruleset(s), including the failed ruleset, have not been migrated. **Solution:** Check the RiskFort logs for details and ensure appropriate action is taken based on these logs. Remove the cyclic dependency of the failed ruleset and migrate it. Also migrate the remaining rulesets that were not migrated because of the failed ruleset.
7655		The value could not be converted to Upper case.	**Possible Cause:** The input given cannot be converted to uppercase. **Solution:** Check the input.

Table D-3. Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
7656	0	The User Profile information could not be retrieved.	**Possible Cause:** UDS is not up and running. **Solution:** Check if UDS is deployed properly and is up and running. See the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for more information to do so. **Possible Cause:** The user does not exist in the RiskFort system. **Solution:** Add the user to the RiskFort system.
7657		The Location and Connection information was not found in the database.	**Possible Cause:** The Quova data has not been uploaded in the RiskFort database. **Solution:** Upload the Quova data into the RiskFort database by using the `arrfupload.exe` tool. See the *Arcot RiskFort 2.2.6 Administration Guide* for more information on this.
7658		The Exception User does not exist in the system.	**Possible Cause:** The specified user is not an Exception User. **Solution:** Ensure that you have provided the correct details for the user, or add the user to the Exception User List before proceeding.

Appendix E
Input Data Validations

To ensure that the system does not process invalid data, to enforce business rules, and to ensure that user input is compatible with internal structures and schemas, RiskFort Server validates the data that it receives from the APIs. Table E-1 explains the criteria that the RiskFort Server uses to validate this input data.

Note: Attribute length mentioned in the following table corresponds to the character length. Attribute ID is referred to as `paramName` in the Java APIs.

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
User name	USER_NAME	User name is non-empty.
		Length is between 1 and 256 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Org name	ORG_NAME	Organization name is non-empty.
		Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Display Org Name	DISPLAY_ORG_NAME	Organization name is non-empty.
		Length is between 1 and 1024 characters.
		Does not contain invalid characters.
Start time	START_TIME	Is non-empty.
End time	END_TIME	Is non-empty.
Duration	DURATION	The dates specifies cannot be in the past.
Create Time	CREATE_TIME	Is non-empty.
		Is less than or equal to the current time.
Last Modified time	LAST_MODIFIED_TIME	Is non-empty.
		Is less than or equal to the current time.

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Config name	CONFIG_NAME	Configuration name is non-empty.
		Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Channel name	CHANNEL_NAME	Channel name is non-empty.
		Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Config state	CONFIG_STATE	Length is between 1 and 5 characters.
Config state for AdminWS	CONFIG_STATE_WS	Length is not more than 2 characters; though can be 0.
Country name	COUNTRY_NAME	Country name is non-empty.
		Length is between 0 and 50 characters.
		Does not contain invalid characters (ASCII 0-31).
Country code	COUNTRY_CODE	Country code is non-empty.
		Length is between 1 and 2 characters.
		Can contain numbers, alphabets, underscore, and dot.
Start IP	START_IP	Start IP address is non-empty.
		Length is between 0 and 4294967295 characters.
		Follows the IP address format.
End IP	END_IP	End IP address is non-empty.
		Length is between 0 and 4294967295 characters.
		Follows the IP address format.
Mask	MASK	Mask is non-empty.
		Length is between 0 and 4294967295 characters.
		Follows the IP address format.
Start IP	START_IP_STR	Start IP address is non-empty.
		Length is between 7 and 15 characters.
		Follows the IP address format.

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
End IP	END_IP_STR	End IP address is non-empty.
		Length is between 7 and 15 characters.
		Follows the IP address format.
Mask	MASK_STR	Mask is non-empty.
		Length is between 7 and 15 characters.
		Follows the IP address format.
Untrusted IP type	UNTRUSTED_IP_TYPE	Length is between 1 and 65535 characters.
Start IP Filter	START_IP_FILTER	Start IP filter is non-empty.
		Length is between 7 and 15 characters.
		Follows the IP address format.
Source IP Filter	SOURCE_IP_FILTER	Source IP filter is non-empty.
		Length is between 7 and 15 characters.
		Follows the IP address format.
Rule name	RULE_NAME	Rule name is non-empty.
		Length is between 1 and 128 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Rule Mnemonic	RULE_MNEMONIC	Rule mnemonic is non-empty.
		Length is between 1 and 128 characters.
		Does not contain invalid characters, although numbers, alphabets, underscore (_), and hyphen (-) are allowed.
Rule description name	RULE_DESCR_NAME	Rule description name is non-empty.
		Length is between 0 and 256 characters.
		Does not contain invalid characters, although ASCII 0-31 are allowed.
Rule type	RULE_TYPE	Rule type is non-empty.
		Length is between 1 and 128 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Rule library name	LIB_NAME	Rule library name is non-empty.
		Length is between 1 and 128 characters.
		Does not contain invalid characters, although numbers, alphabets, underscore (_), and hyphen (-) are allowed.
Parameter name	RULE_PARAM_NAME	Parameter name is non-empty.
		Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Parameter value	RULE_PARAM_VALUE_STR	Parameter value is non-empty.
		Length is between 1 and 512 characters.
		Does not contain invalid characters (ASCII 0-31).
Parameter value	RULE_PARAM_VALUE_BIN	Parameter value is non-empty.
Parameter type	RULE_PARAM_TYPE	Length is between 1 and 4 characters.
Description	DESCRIPTION	Description is non-empty.
		Length is between 1 and 256 characters.
		Does not contain invalid characters (ASCII 0-31).
Aggregator name	AGGREGATOR_NAME	Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Aggregator ID	AGGREGATOR_ID	--
Score	SCORE	Value is between 1 and 100.
Scoring priority	SCORING_PRIORITY	Value is between 1 and 2147483647.
Execution priority	EXEC_PRIORITY	Value is between 0 and 100000.
Exection enabled	EXECUTIONENABLED	Value must either be 0 or 1.
Scoring enabled	SCORINGENABLED	Value must either be 0 or 1.
Other Org name	OTHERORGNAME	Length is between 1 and 64 characters.
		Does not contain invalid characters.

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Other Config name	OTHERCONFIGNAME	Length is between 1 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Sharing Type	SHARINGTYPE	Value is between 0 and 3.
Callout type	CALLOUT_TYPE	Value is between 0 and 2.
Callout URL	CALLOUT_URL	URL is non-empty.
		Length is between 0 and 150 characters.
		Does not contain invalid characters, although alphabets, number, and + / \ \ # $ % & - _ : . are allowed.
Callout timeout	CALLOUT_TIMEOUT	Value is between 0 and 1000000.
Instance name	INSTANCE_NAME	Length is between 0 and 32 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Protocol Module name	PROTOCOL_MODULE_NAME	Length is between 0 and 128 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Client SSL Trust Store name	CLIENT_SSL_TRUST_STORE_NAME	Length is between 0 and 64 characters.
		Does not contain invalid characters, although ASCII 32-127 are allowed.
Connection timeout	CONNECTION_TIMEOUT	Value is between 0 and 1000000.
Client cert	CLIENT_CERT	--
Client key	CLIENT_KEY	--
Server root CA cert	SERVER_ROOT_CA_CERT	--
Server Private key	SERVER_PRIVATE_KEY	--
Read timeout	READ_TIMEOUT	--
Min connections	MIN_CONNECTION	--
Max connections	MAX_CONNECTION	--

Table E-1. Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Full distinguished name of the certificate	CERT_SUBJECT	--
Issuer name	ISSUER_NAME	--
Server SSL auth	SERVER_AUTH_SSL	--
Client SSL auth	CLIENT_AUTH_SSL	--
Action	TRANS_ACTION	Action is non-empty.
		Length is between 1 and 32 characters.
		Does not contain invalid characters (ASCII 0-31).
Association name	ASSOC_NAME	Association name is non-empty.
		Length is between 1 and 32 characters.
		Does not contain invalid characters (ASCII 0-31).

Appendix F
RiskFort Logging

To effectively manage the communication between RiskFort Server and your application, it is necessary to get information about the activity and performance of the Server and other components, as well as any problems that might have occurred.

This appendix describes the various log files supported by RiskFort, the severity levels that you will see in these files, and the formats of these log files. It covers the following topics:

About the Log Files

The RiskFort log files can be categorized as:

- Installation Log File
- Transaction Log Files
- UDS Log File
- Administration Console Log File

All logging-related parameters (specified under the `[arcot/riskfort/logger]` section) are controlled by the `riskfortserver.ini` file. These logging configuration options include:

- **Specifying log file name and path:** RiskFort enables you to specify the directory for writing the log files and storing the backup log files. Specifying the diagnostic logging directory allows administrators to manage system and network resources.
- **Log file size:** The maximum number of bytes the log file can contain. When the log files reach this size, a new file with the specified name is created and the old file is moved to the backup directory.

- **Using log file archiving:** As RiskFort components run and generate diagnostic messages, the size of the log files increases. If you allow the log files to keep increasing in size, then the administrator must monitor and clean up the log files manually. RiskFort enables you to specify configuration options that limit how much log file data is collected and saved. RiskFort lets you specify the configuration option to control the size of diagnostic logging files. This lets you determine a maximum size for the log files. When the maximum size is reached, older log information is moved to the backup file before the newer log information is saved.
- **Setting logging levels:** RiskFort also allows you to configure logging levels. By configuring logging levels, the number of messages saved to diagnostic log files can be reduced. For example, you can set the logging level so that the system only reports and saves critical messages. See "Supported Severity Levels" on page F-157 for more information on the supported log levels.
- **Specifying time zone information:** RiskFort enables you to either use the local time zone for time stamping the logged information or use GMT for the same.

Installation Log File

When you install RiskFort, the installer records all actions that it performs in the `Arcot_RiskFort_InstallLog.log` file. The information in this file is very useful in identifying the source of the problems if the RiskFort installation did not complete successfully.

The default location of this file is:

Windows:

`<install_location>\Arcot Systems\logs\`

UNIX-Based:

`<install_location>/arcot/logs/`

Transaction Log Files

The transaction logs consist of:

- RiskFort Server Log
- Case Management Server Log File

RiskFort Server Log

RiskFort records all requests processed by the server and related actions in the `arcotriskfort.log` file. The default location of this file is:

Windows:

`<install_location>\Arcot Systems\logs\`

UNIX-Based:

`<install_location>/arcot/logs/`

Note: You cannot use the RiskFort logger to configure your application's logs. You can access these logs by using the tool used by the third-party application server (such as Apache Tomcat or IBM Websphere) that is hosting your application.

All logging-related parameters (specified under the `[arcot/riskfort/logger]` section) are controlled by the `riskfortserver.ini` file, which is available in the `conf` folder in `ARCOT_HOME`.

Case Management Server Log File

When you deploy the Case Management Server module and subsequently start it, the details of all its actions and processed requests are recorded in the `arcotriskfortcasemgmtserver.log` file. The default location of this file is:

Windows:

`<install_location>\Arcot Systems\logs\`

UNIX-Based:

`<install_location>/arcot/logs/`

Note: You cannot use the RiskFort logger to configure your application's logs. You can access these logs by using the tool used by the third-party application server (such as Apache Tomcat or IBM Websphere) that is hosting your application.

All logging-related parameters (specified under the `[arcot/riskfortcasemgmtserver/logger]` section) are controlled by the `riskfortcasemgmtserver.ini` file, which is available in the `conf` folder in `ARCOT_HOME`.

UDS Log File

All User Data Service (UDS) information and actions are recorded in the `arcotuds.log` file. This information includes:

- UDS database connectivity information
- UDS database configuration information
- UDS instance information and the actions performed by this instance

The information in this file is very useful in identifying the source of the problems if the Administration Console could not connect to the UDS instance. The default location of this file is:

Windows:

`<install_location>\Arcot Systems\logs\`

UNIX-Based:

`<install_location>/arcot/logs/`

The parameters that control logging in this files can be configured by using the `udsserver.ini` file, which is available in the `conf` folder in `ARCOT_HOME`.

In addition to the logging level, log file name and path, the maximum file size (in bytes), and archiving information, you can also control the layout of the logging pattern for UDS by specifying the appropriate values for `log4j.appender.debuglog.layout.ConversionPattern`.

See section, "Format of UDS and Administration Console Log Files" on page F-156 for the details of the default format used in the file.

Administration Console Log File

When you deploy the Administration Console and subsequently start it, the details of all its actions and processed requests are recorded in the `arcotadmin.log` file. This information includes:

- Database connectivity information
- Database configuration information
- Instance information and the actions performed by this instance
- UDS configuration information
- Other Administration Console information specified by the Master Administrator, such as cache refresh

The information in this file is very useful in identifying the source of the problems if the Administration Console does not start up. The default location of this file is:

Windows:

`<install_location>\Arcot Systems\logs\`

UNIX-Based:

`<install_location>/arcot/logs/`

The parameters that control logging in this files can be configured by using the `adminserver.ini` file, which is available in the `conf` folder in `ARCOT_HOME`.

In addition to the logging level, log file name and path, the maximum log file size (in bytes), log file archiving information, you can also control the layout of the logging pattern for the console by specifying the appropriate values for `log4j.appender.debuglog.layout.ConversionPattern`.

See section, "Format of UDS and Administration Console Log Files" on page F-156 for the details of the default format used in the file.

Format of the RiskFort Server and Case Management Server Log Files

Table F-1 describes the format of the entries in the RiskFort logger, `arcotriskfort.log`, as discussed in the section, "RiskFort Server Log" on page F-153.

Table F-1. RiskFort Logging Format

Column	Description
Time Stamp	Time the entry was logged, translated to the specified time zone. The format of logging this information is: `www mmm dd HH:MM:SS.mis yy timezone` In the preceding format: • `www` represents weekday. • `mis` represents milliseconds. • `timezone` represents the time zone you specified in the `riskfortserver.ini` file.
Log Level (or Severity)	The severity level of the logged entry. See "Supported Severity Levels" for detailed information.

Table F-1. RiskFort Logging Format

Column	Description
Process ID (`pid`)	The ID of the process that logged the entry.
Thread ID (`tid`)	The ID of the thread that logged the entry.
Transaction ID	The ID of the transaction that logged the entry.
Module ID (Deprecated)	The ID of the RiskFort module (such as Rules Engine or Scoring Engine) that logged the entry. **Note:** This is a deprecated entry and always shows `0`.
Message	The message logged by the Server in the free-flowing format. **Note:** The granularity of this message depends on the Log Level that you set in `riskfortserver.ini`.

Format of UDS and Administration Console Log Files

Table F-2 describes the format of the entries in the following loggers:

- `arcotuds.log` (UDS Log File)
- `arcotadmin.log` (Administration Console Log File)

Table F-2. UDS and Administration Console Logging Format

Column	Associated Pattern (In the Log File)	Description
Time Stamp	`%d{yyyy-MM-dd hh:mm:ss,SSS z} :`	The time when the entry was logged. This entry uses the application server time zone. The format of logging this information is: `yyyy-MM-dd hh:mm:ss,SSS z` Here: • `SSS` represents milliseconds. • `z` represents the time zone.
Thread ID	`[%t] :`	The ID of the thread that logged the entry.
Log Level (or Severity)	`%-5p :`	The severity level of the logged entry. See "Supported Severity Levels" for more information.

Table F-2. UDS and Administration Console Logging Format

Column	Associated Pattern (In the Log File)	Description
Logger Class	`%-5c{3}(%L) :`	The name of the logger that made the log request.
Message	`%m%n :`	The message logged by the Server in the log file in the free-flowing format. **Note:** The granularity of the message depends on the Log Level that you set in the log file.

Refer to the following URL for customizing the `PatternLayout` parameter in the UDS and Administration Console log files:

http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

Supported Severity Levels

A *log level* (or *severity level*) enables you to specify the level of detail of the information stored in the RiskFort logs. This also enables you to control the rate at which the log file will grow.

Table F-3 describes the log levels that you see in all log files, in the *decreasing* order of severity.

Table F-3. RiskFort Log Levels (in Decreasing Order of Severity)

Log Level		Description
0	`FATAL`	Use this log level for serious, non-recoverable errors that can cause the abrupt termination of the RiskFort service.
1	`WARNING`	Use this log level for undesirable run-time exceptions, potentially harmful situations, and recoverable problems that are not yet `FATAL`.
2	`INFO`	Use this log level for capturing information on run-time events. In other words, this information highlights the progress of the application, which might include changes in: • Server state, such as start, stop, and restart. • Server properties. • State of services. • State of a processes on the Server.
3	`LOW DETAIL`	Use this log level for logging detailed information for debugging purposes. This might include process tracing and changes in Server states.

Note: When you specify a log level, messages from all other levels of *higher* significance are reported as well. For example if the `LogLevel` is specified as `3`, then messages with log levels of `FATAL`, `WARNING`, and `INFO` level are also captured.

Sample Entries for Each Log Level

The following subsections show a few sample entries (based on the Log Level) in the **RiskFort log file.**

FATAL

```
May 27 18:31:01.585 2010 GMT FATAL: pid 4756 tid 5152: 0: 0: Cannot
continue due to ARRF_LIB_init failure, SHUTTING DOWN
```

WARNING

```
May 24 14:47:39.756 2010 GMT WARNING: pid 5232 tid 5576: 0: 110000:
EVALHTTPCALLOUT : Transport Exception : create: No Transports Available
```

INFO

```
May 24 14:41:43.758 2010 GMT INFO: pid 3492 tid 4904: 0: 109002: Error
in ArPFExtRuleSetEval::evaluate Could not get user context (two
parallel requests)

May 25 10:01:28.131 2010 GMT WARNING: pid 1048 tid 3104: 8: 0: Error in
ArRFCaseStatus::startInit: No data found
```

DETAIL

```
May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Entering USERRISKEVALVELOCITY Rule Evaluation
function


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE: VELOCITY_DURATION=[60],
VELOCITY_DURATION_UNIT=[MINUTES],               VELOCITY_TRANSACTION_COUNT=[5]


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Entering UserRiskEvalVelocityRule
durationToTimeConvertor

May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Exiting UserRiskEvalVelocityRule
durationToTimeConvertor


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Entering UserRiskEvalVelocityRule
callUserEvalVelocityRule


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Entering
ArUserRiskEvalVelocityDBO::decisionLogicForUserVelocity


May 24 14:52:01.219 2010 GMT INFO: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Entering decisionLogicForUserVelocity


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Exiting
ArUserRiskEvalVelocityDBO::decisionLogicForUserVelocity


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Exiting UserRiskEvalVelocityRule
callUserEvalVelocityRule


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : USERRISKEVALVELOCITY.RESULT=[0]
```

```
May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : USERRISKEVALVELOCITY.DETAIL=[RESULT=0;TCOUNT=2;
ACT=mection]


May 24 14:52:01.219 2010 GMT LOW: pid 2132 tid 1356: 0: 111004:
USERRISKEVALVELOCITYRULE : Exiting USERRISKEVALVELOCITY Rule Evaluation
function
```

Appendix G
RiskFort Sample Application

This version of RiskFort is shipped with Sample Application that serves as a “template" of simple Java primitives (code) that demonstrate the usage of RiskFort Java APIs and how your application can be integrated with RiskFort. In this manner, Sample Application serves to standardize the integration process between RiskFort and your application.

This appendix contains the following sections:

Understanding the Sample Application

To deploy RiskFort in your environment, you need to integrate its APIs with your online application. Sample Application uses RiskFort Java APIs to demonstrate the most common functionality of RiskFort, discussed in detail in the following sections:

- “Performing Risk Evaluation” on page G-167
- “Creating Users” on page G-177

	**Important:** Sample Application *must not* be used in your production environment. Arcot strongly recommends that you build your own Web application by using Sample Application *only* as a code-reference. In a production environment, Sample Application can only be used to verify if RiskFort was installed successfully, and if is able to perform risk-evaluation operations.

Sample Application Components

Sample Application constitutes the following:

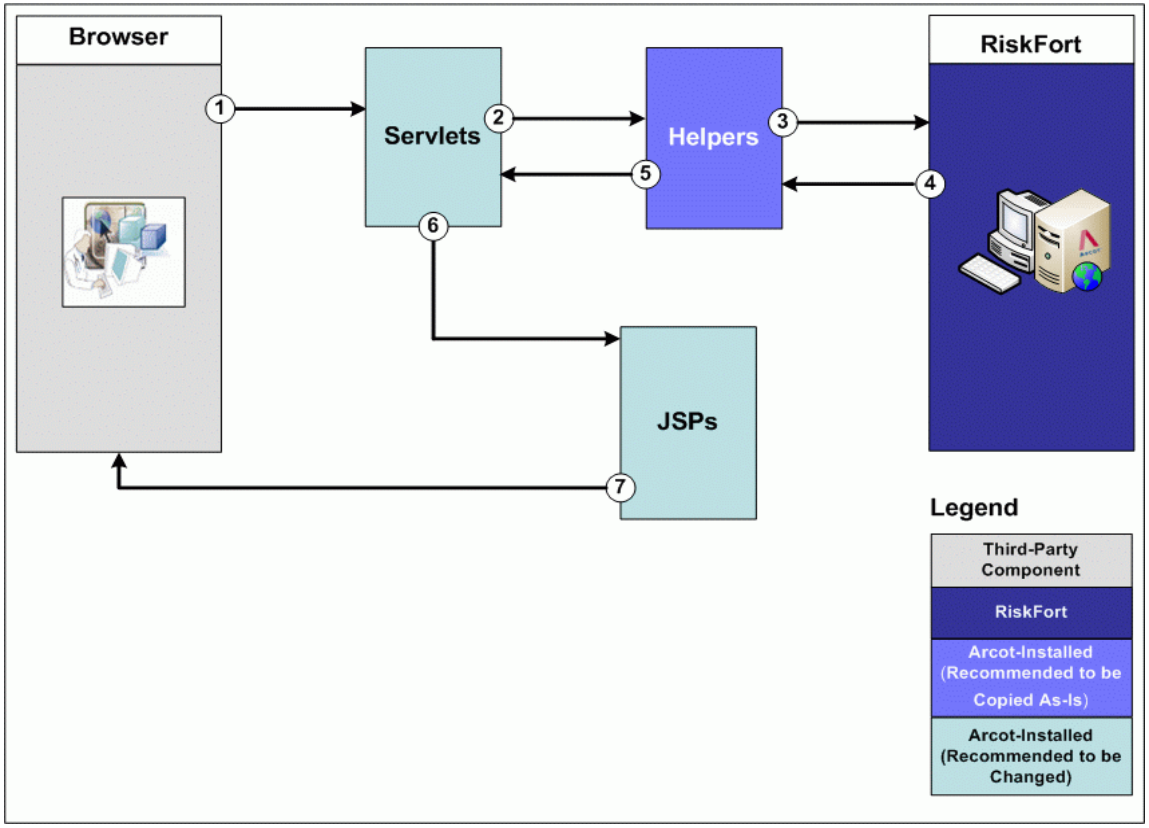
- **Servlets:** Platform-independent server-side modules that can be used to add new functionality to a Web server. (Servlets are an equivalent of an applets, however, unlike applets, they do not have a user interface. They just enable you to access the existing business logic by the help of user interface provided by JSPs.
- **Helpers:** Classes that provide the extra functionality that is not a part of the class that makes use of them. As a result, helper classes help making the code maintainable and reusable.

  For example, if a class needs to display the value of a number, it could just display it raw, or alternately format it neatly (comma-separated format, for example). To do the formatting instead, the class can make use of another class, which is referred to as a helper class.
- **JSPs:** Java Server Pages (JSPs) that enable you to create and serve dynamic Web content that is server- and platform-independent.

These Sample Application components are organized in a *model-view-controller* (MVC) framework, which separates the user interface of your application from the underlying business logic. As a result, it is much easier to modify either the user interface of the application or the business rules or both without affecting the other.

Figure G-1 illustrates the interaction between the components of Sample Application (*after* a user is successfully authenticated by your application.)

Figure G-1 Interaction Between the Components of the Sample Application



As illustrated in Figure G-1, the flow of information between the various components of Sample Application is as follows:

1. User initiates a transaction using the browser window.

	Note: This step assumes that the end user is already successfully logged in after being authenticated by your application.

2. The *servlet* processes the request and invokes the corresponding *helper* function so that it can, in turn, invoke the required RiskFort SDK.
3. The helper function calls the appropriate API to interact with RiskFort Server by passing the inputs forwarded by the servlet in the preceding step.

4. RiskFort Server assesses the input and returns a risk score and an advice. In each case, an object of the relevant type is returned.

   If the assessment failed, an exception is generated.

5. The helper function returns the object created by RiskFort SDK along with the response to the corresponding servlet.

   In case of a failure, the helper function catches all exceptions and displays a meaningful error message.

6. The servlet receives the response from the helper function, sets the corresponding values in request attribute, and forwards it to a JSP.

7. The JSP parses the request attribute set by the servlet, generates an HTTP response, and forwards it to the browser.

   This response is then displayed to the user.

Sample Application Recommendations

In this integration model, Arcot strongly recommends that:

- You can use the helper functions *without* any modifications.
- You replace all Sample Application-provided JSPs with your own to control the look and feel of the user interface of the application.

Note: Read the API documentation *before* changing the code.

- You replace all Sample Application-provided servlets with your own controller logic.
- You *do not use* Sample Application in your production environment. Instead, it is recommended that build your own Web application *by using* Sample Application only as a code-reference.

Installing and Configuring the Sample Application

You can install the Sample Application if you select the **Complete** option while running the RiskFort installer. When you do so, the Sample Application is also installed along with other RiskFort components.

As a part of **Custom** installation, you must select the **Risk Evaluation** option on the Choose Install Set screen to install the Sample Application.

Book: For detailed information on prerequisites of the Sample Application and its deployment, refer to the *Arcot RiskFort 2.2.6 Installation and Deployment Guide*.

On Windows

To install (and later configure) RiskFort and the Sample Application on Windows successfully, the user account that you plan to use for installation *must* belong to the `Administrators` group. Else, some critical steps in the installation, such as DNS creation and configuration and RiskFort service creation, will not go through successfully, though the installation might complete without any errors.

On UNIX-Based Platforms

The installation UNIX-based platforms is not restricted to the `root` account. You can install as a non-root user.

Configuring Sample Application

You can deploy Sample Application in any of the scenarios discussed in the following sub-sections.

Sample Application and RiskFort on the Same System

If Sample Application and RiskFort Server are running on the same system, you *do not* have to perform any steps to configure Sample Application.

However if your RiskFort Server uses a port other than the default (`7680`), then the only deployment steps you need to perform is to change the port number. To do so:

1. Ensure that the `riskfort-2.2.6-sample-application.war` is deployed on the application server.

Book: To ensure that Sample Application is deployed correctly, refer to the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for details.

2. Navigate to the location where the WAR file is deployed and open `riskfort.risk-evaluation.properties` file.

Note: The deployment procedure (and `App_Home`) will depend on the application server that you are using.

Refer to your application server documentation for detailed instructions.

On Apache Tomcat, `riskfort.risk-evaluation.properties` is typically available at:

`<App_Home>\webapps\riskfort-2.2.6-sample-application\WEB-INF\classes\properties\`**`riskfort.risk-evaluation.properties`**

3. Ensure that the value of `HOST.1 is localhost`.
4. Change the value of `PORT.1`.

   `PORT.1` represents the port on which RiskFort Server is listening to the incoming requests. For example, `PORT.1=7680`.

Sample Application and RiskFort on Different Systems

If Sample Application and RiskFort Server are running on different systems, then you must complete the following steps to configure Sample Application:

1. Ensure that the `riskfort-2.2.6-sample-application.war` is deployed on the application server.

Book: To ensure that Sample Application is deployed correctly, refer to the *Arcot RiskFort 2.2.6 Installation and Deployment Guide* for details.

2. Navigate to the location where the WAR file is deployed and open `riskfort.risk-evaluation.properties` file.

Note: The deployment procedure (and `App_Home`) will depend on the application server that you are using.

Refer to your application server documentation for detailed instructions.

On Apache Tomcat, `riskfort.risk-evaluation.properties` is typically available at:

`<App_Home>\webapps\riskfort-2.2.6-sample-application\WEB-INF\classes\properties\`**`riskfort.risk-evaluation.properties`**

3. Change the value of `HOST.1`.

   `HOST.1` represents the host name or IP address of RiskFort Server. For example, `HOST.1=10.150.1.111`.

4. Change the value of `PORT.1`.

   `PORT.1` represents the port on which RiskFort Server is listening to the incoming requests. For example, `PORT.1=7680`.

5. **(Optional, if RiskFort Server not running in SSL mode)** Perform this step only if you want to secure the communication between Sample Application and the RiskFort Server over SSL.

   In this case, change the value of `TRANSPORT_TYPE` to `SSL`:

   `TRANSPORT_TYPE=SSL`

6. Specify server CA certificate location for `CA_CERT_FILE=<Server_CA_certificate_location>`.

   Important: The certificate *must* be in `PEM` format.

   For example, `CA_CERT_FILE=C:\certs\riskfort_ca.pem`

7. Save the changes and close the open files.
8. Restart the application server on which the Sample Application is running.

With the completion of these steps, your Sample Application deployment is also complete. You can now start using it to understand the API workflow for each supported operation, as demonstrated in the following sections.

Performing Risk Evaluation

RiskFort Sample Application demonstrates some of these aspects of the RiskFort risk-evaluation process. It covers the following:

- Performing Risk Evaluation on Gathered Information
- Editing the Gathered User Information

- API Workflow and Reference

Performing Risk Evaluation on Gathered Information

Sample Application demonstrates how APIs are used to perform risk evaluation on the gathered data. To view the demonstration, perform the following steps:

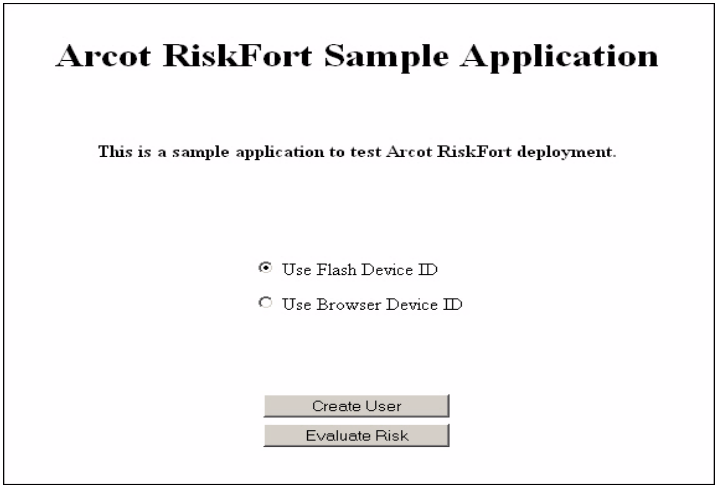
1. Access the Sample Application URL in a browser window. This URL typically is:

   http://<app_server_host_name>:<port_number>/riskfort-2.2.6-sample-application/index.jsp

   In the preceding URL, <app_server_host_name> represents the host name or IP address of the application server on which Sample Application has been deployed and <port_number> represents the port number at which Sample Application is available.

   The RiskFort Sample Application landing page, as shown in the Figure G-2 appears.

   Figure G-2 RiskFort Sample Application: Landing Page

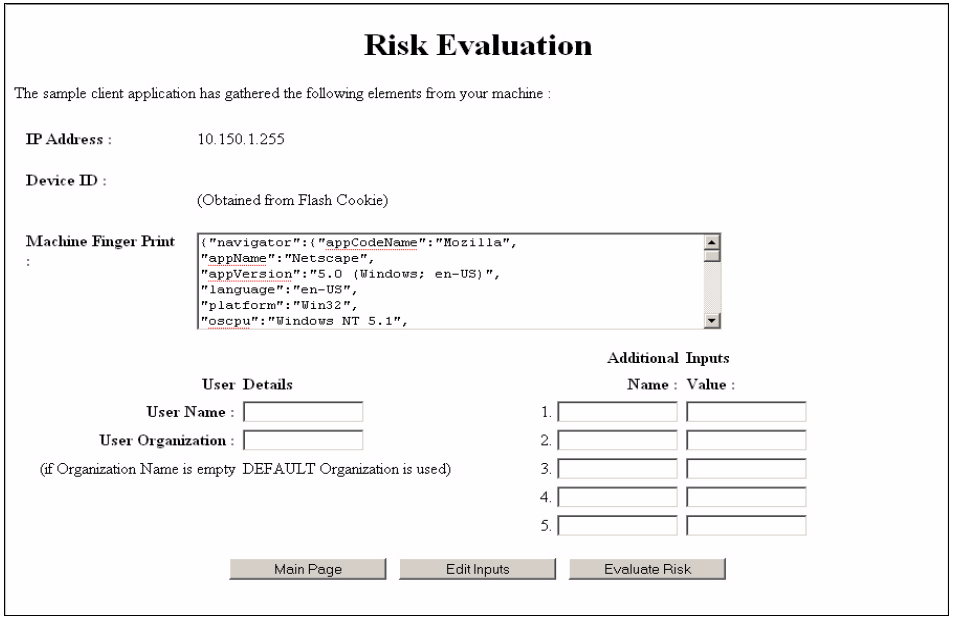
   

2. Select the **Use Flash Device ID** or **Use Browser Device ID** option based on your Device ID storage preference.

   RiskFort uses this stored Device ID as a parameter for risk evaluation.

3. Click **Risk Evaluation** to open the Risk Evaluation on My Default Profile page ().

Figure G-3 Risk Evaluation Input Page



4. On the page, specify:
   - The name of the user who is being evaluated for risk in the **User Name** field.

> **Note:** You can also specify the name of a user who is not yet enrolled with Sample Application, as discussed in "Performing Risk Evaluation" on page G-167.

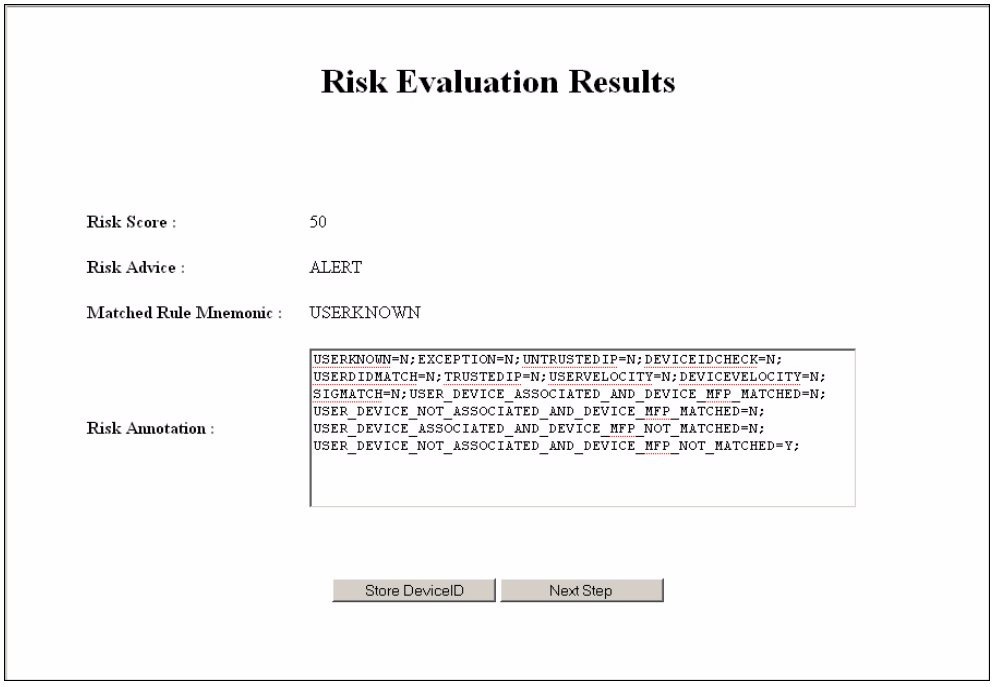
   - (Optional) The name of the organization to which the user belongs in the **User Organization** field.

> **Note:** If you leave the **User Organization** field empty, the value is automatically set to `DEFAULTORG`.

   - (Optional) The corresponding **Name** and **Value** fields with **Additional Inputs**, such as locale information and transaction amount for the current transaction.
5. Click **Evaluate Risk** to generate the Risk Score and Risk Advice for the specified user.

The Risk Evaluation Results page appears. If the user is not enrolled with RiskFort, then the advice is typically, ALERT, as shown in Figure G-4. This advice changes when you create the user in RiskFort database, as discussed in section, "Creating Users" on page G-177 later in this appendix.

Figure G-4 Risk Evaluation Result



6. Click **Store DeviceID** to store the specified type of Device ID information (Step 2) on the local computer of the user:

   - Typically, the Flash SharedObject (or token) is stored in an appropriate subdirectory in the following location:

     - **Microsoft Windows:**

       `C:\Documents and Settings\<user_profile>\Application Data\Macromedia\Flash Player\#SharedObjects\`

     - **UNIX Platforms:**

       `/<user_home>/macromedia/flash/#SharedObjects/`

   - If you selected to use the Browser cookie, then based on the browser that you are using, this cookie is available in the following location:

Windows Internet Explorer:

- **Microsoft Windows:**

```
C:\Documents and Settings\meeta\Local Settings\Temporary Internet
Files\
```

Mozilla Firefox:

- **Microsoft Windows:**

```
C:\Documents and Settings\<user_profile>\Application
Data\Mozilla\Firefox\Profiles\<random_name>.default
```

- **UNIX Platforms:**

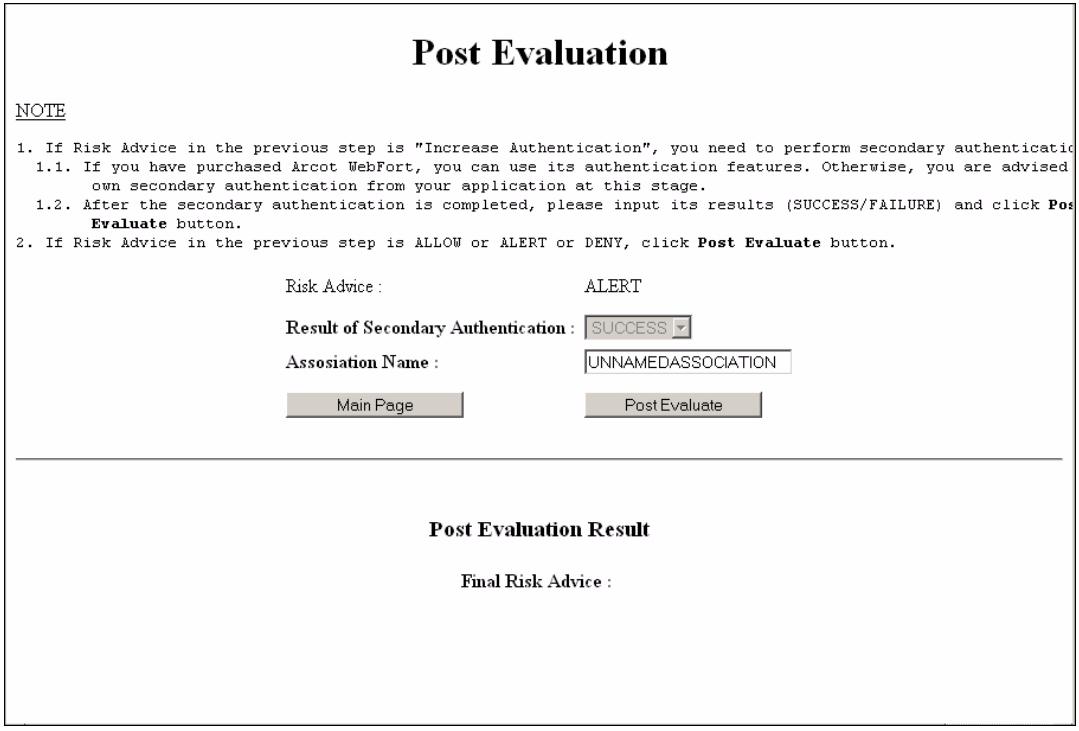
```
/<user_home>/mozilla/firefox/profiles/<random_name>.default/
```

7. Click **Next Step** to perform post evaluation for the specified user.

   The Post Evaluation page (Figure G-5) appears.

Important: As indicated on the page, if the advice on the previous page was `INCREASEAUTH`, then Arcot strongly recommends that you plug-in your secondary authentication mechanism, so that the user performs another authentication before performing RiskFort post-evaluation steps.

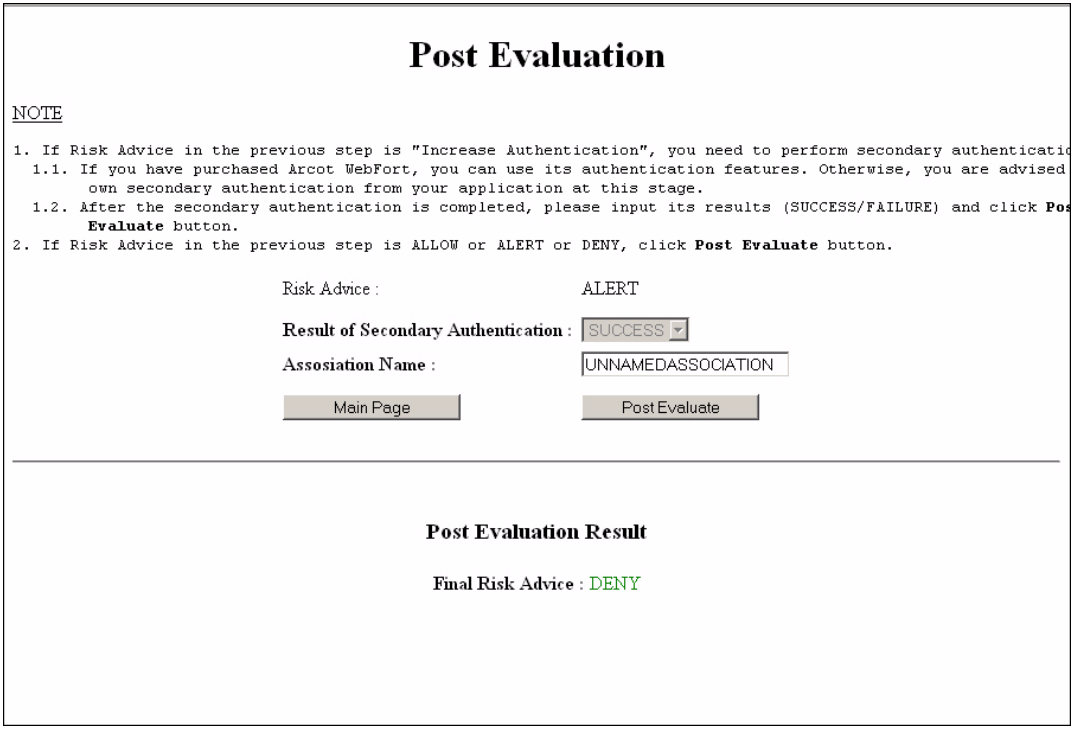
Figure G-5 Post Evaluation Page



8. Click **Post Evaluate** to generate and display the final Risk Advice for your application.

   The final result is displayed in the Post Evaluation Results frame of the page (Figure G-6).

Figure G-6 Post Evaluation Result (Lower Frame of the Page)



Editing the Gathered User Information

Sample Application also allows you to edit the collected Device ID information of an existing user. This enables you to simulate real-life situations a user might run in to. For example, if you change the IP address of an enrolled user, then you can simulate that the user is logging in from a different location.

For demonstration purposes, the data that you can edit by using Sample Application is:

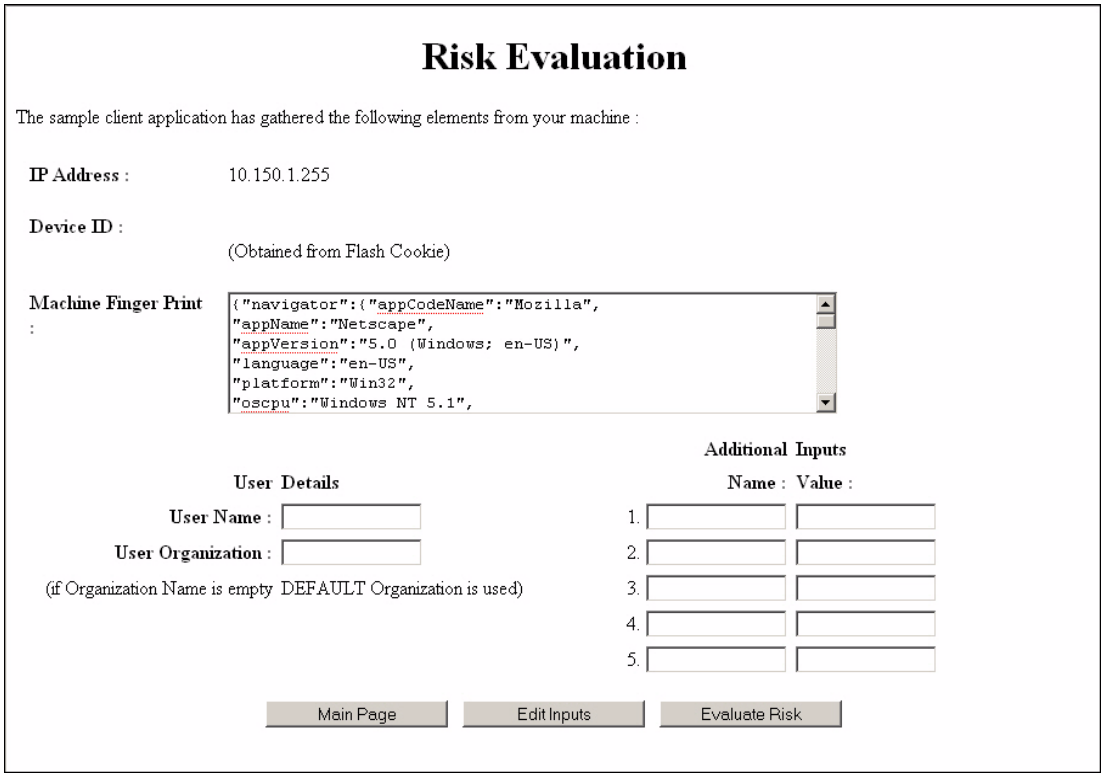
- IP Address
- Device ID

After a user's information is updated, they must undergo the risk-evaluation steps, as discussed in Step 5 on page G-169 through Step 8 on page G-172. In such cases, RiskFort typically generates the `INCREASEAUTH` advice.

To edit a user's information by using Sample Application, perform the following tasks:

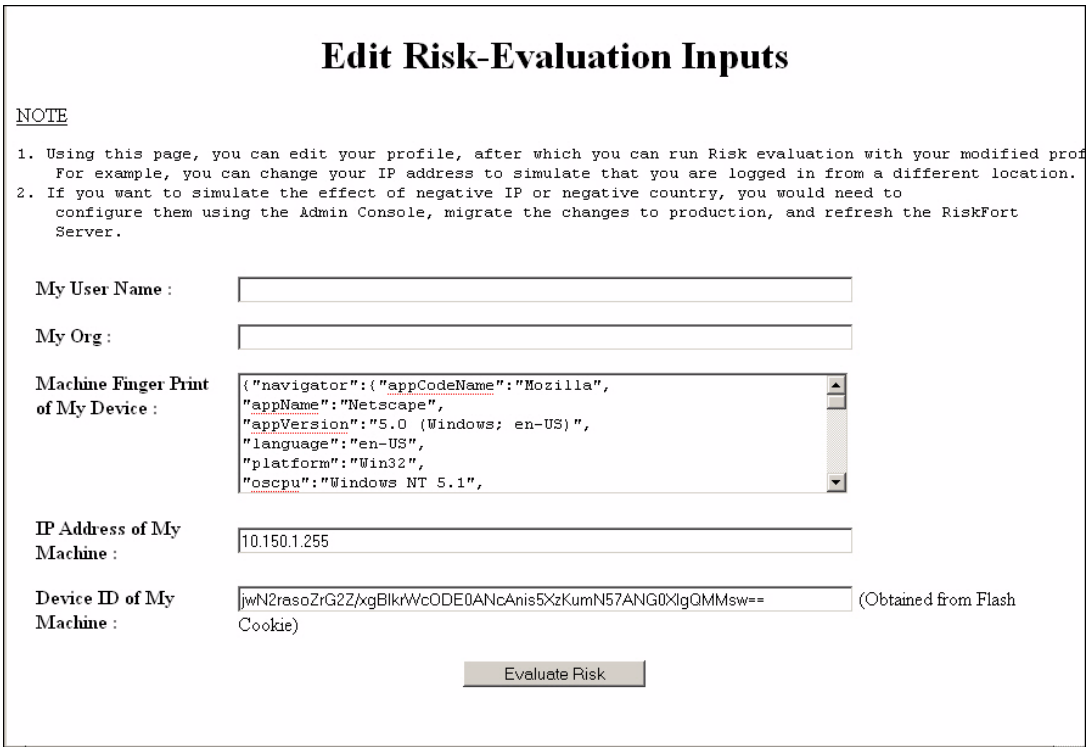
1. On the RiskFort Sample Application page (Figure G-2), Click **Risk Evaluation** to open the Risk Evaluation input page (Figure G-7).

Figure G-7 Risk Evaluation Input Page



2. Specify the **User Name**, and if required the **User organization**.
3. Click **Edit Inputs** to open the Edit Risk-Evaluation Inputs page (Figure G-8).

Figure G-8 Edit Risk-Evaluation Inputs Page



4. Change the IP address in the **IP Address of My Machine** field or the Device ID in the **Device ID of My Machine** field, or both.
5. Click **Evaluate Risk** to generate the Risk Score and Risk Advice for the specified user.

   The Risk Evaluation Results page (Figure G-4) appears.
6. Click **Store DeviceID** to store the specified type of Device ID information on the local computer of the user.
7. Click **Next Step** to perform post evaluation for the specified user.

   The Post Evaluation page appears.
8. Specify the result of secondary authentication (to simulate the scenario where RiskFort receives the result of this additional authentication) in the **Result of Secondary Authentication** field.
9. Click **Post Evaluate** to generate and display the final Risk Advice for your application.

The final result is displayed in the Post Evaluation Results frame of the page.

API Workflow and Reference

When you use Sample Application for risk evaluation:

1. The `ArRFInitHandler.class` of the `com.arcot.riskfort.sampleapp.initialize` package is invoked.

   After the initialization is complete, RiskFort is ready to serve the requests.

2. `RiskFactory`, which is responsible for RiskFort-related operations, is invoked by the `ArRFEvaluateHelper.class` in the `com.arcot.riskfort.sampleapp.helpers` package.

   When the `evaluateRisk()` method in `ArRFEvaluateHelper.class` is called, the `RiskXActionAPI` interface is invoked.

3. To perform **risk evaluation**, your application's servlet, .jsp, or any other calling file must call the `evaluateRisk()` method of the `ArRFEvaluateHelper` class.

   This method returns the `RiskAssessment` object as response, which contains `riskScore`, `riskAdvice`, `deviceID`, and related information to your application.

	**Important:** Based on the generated risk score and advice at this stage, your application must provide logic to perform the required action, such as forward the transaction request to a CSR or force the user to perform additional authentication.

4. To perform post evaluation, your application's servlet, .jsp, or any other calling file must call the `postEvaluateHelper()` method of the `ArRFEvaluateHelper` class.

	**Important:** The `postEvaluateHelper()` method must be called only after the `evaluateRisk()` method was executed.

   This method accepts `CallerID`, `RiskAssessment` object (returned by the `evaluateRisk()` function), and result of your secondary authentication, if any, (`secondaryAuthenticationStatus`), and association name (optional) as input and returns to your application the `postEvaluateResponseObj` object, which contains the final advice.

The class and method required in the preceding workflow are described in Table G-1.

Table G-1. Main APIs Required for Risk Evaluation Through Sample Application

API	Description
Class: `ArRFEvaluateHelper`	The helper class that contains the methods required for risk evaluation and post evaluation.
`Method:` `evaluateRisk()`	The method that generates the Risk Score and Risk Advice.
`Method:` `postEvaluateHelper()`	The method that generates the final Risk Advice. This advice is a Boolean value. In case of `True`, the advice is `ALLOW`, while in case of `False`, the advice is to not `ALLOW` the transaction. This method must be called only *after* `evaluateRisk()` has been executed.

Creating Users

Creating a new user in RiskFort is a one-time operation, which is performed only when a new user is to be added to RiskFort. This user typically is an existing user of your application accessing RiskFort for the first time.

This section describes how the Sample Application demonstrates the creation of a user and then explains the API workflow for the same:

- Creating a User
- Performing Risk Evaluation for the User that You Created
- API Workflow and Reference

Creating a User

Sample Application demonstrates how APIs are used to create a user in RiskFort. To view the demonstration, perform the following steps:

1. Access the Sample Application URL in a browser window. This URL typically is:

   http://<app_server_host_name>:<port_number>/riskfort-2.2.6-sample-application/index.jsp

   In the preceding URL, <*app_server_host_name*> represents the host name or IP address of the application server on which Sample Application has been deployed and <*port_number*> represents the port number at which Sample Application is available.

The RiskFort Sample Application page (Figure G-2) appears.

2. Click **Create User** to create a new user in RiskFort database.

> **Note:** Creating a user in RiskFort does not depend on any cookie type. As a result, you do not need to specify a Device ID type (Flash or Browser, as shown in the preceding figure) when creating a user.

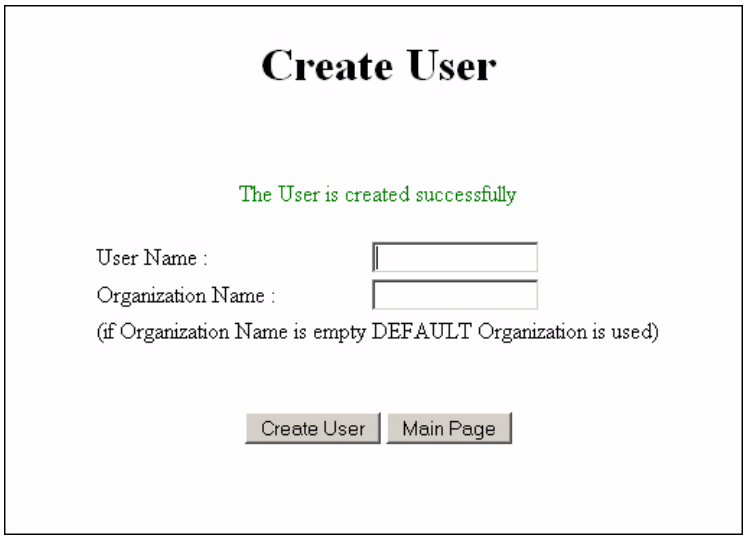
The Create User page (Figure G-9) appears.

Figure G-9 Create User Page

3. Specify the **User Name**, **Organization Name**, and click **Create User**.

   If the specified user was created successfully, the success message, as shown in, appears.

Figure G-10 Create User Page: Success Message

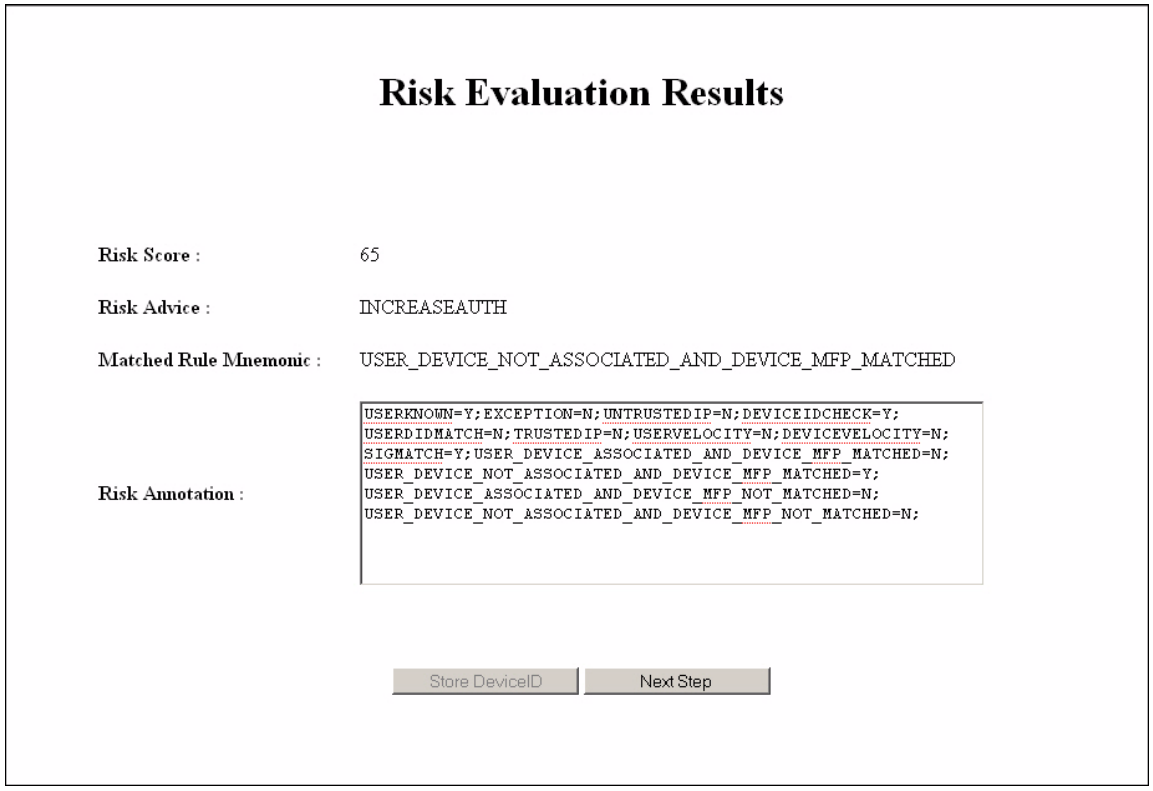


4. Click **Create User** if you want to create another user, or click **Main Page** to return to the starting page (`index.html`) of Sample Application.

Performing Risk Evaluation for the User that You Created

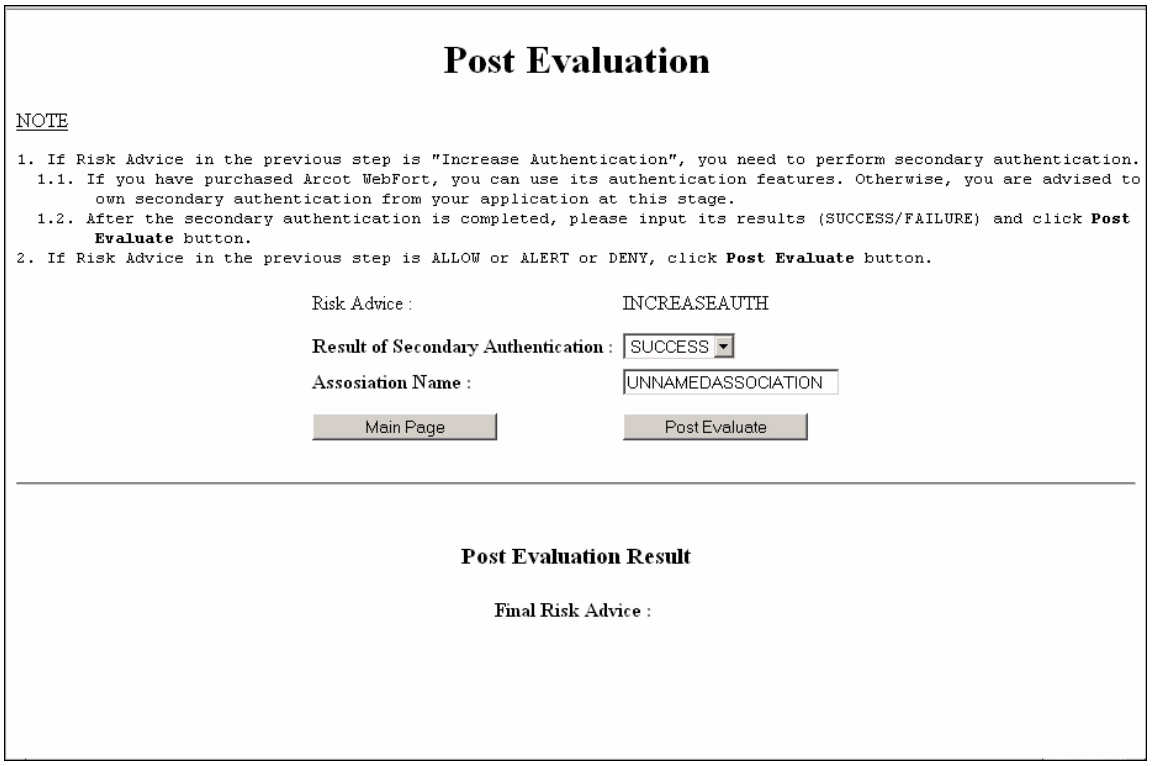
If now you perform risk evaluation (Step 2 on page G-168 through Step 5 on page G-169, as mentioned in section "Performing Risk Evaluation on Gathered Information" on page G-168) for the user that you just created, you will see the **Risk Score** and **Risk Advice** values as `65` and `INCREASEAUTH`, respectively, on the Risk Evaluation Results page (Figure G-11).

Figure G-11 Risk Evaluation Result: After User was Created



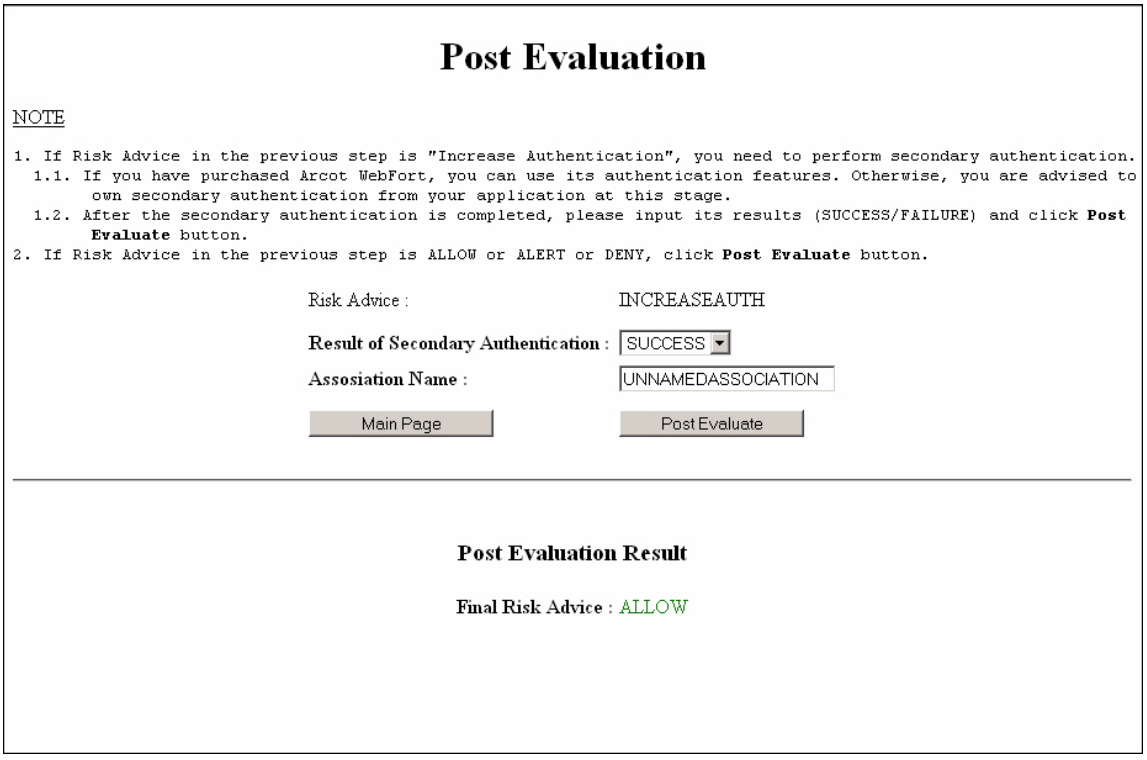
When you click **Next Step** to perform post evaluation, you will see the Post Evaluation page as follows (Figure G-12).

Figure G-12 Risk Post Evaluation Result: After User was Created



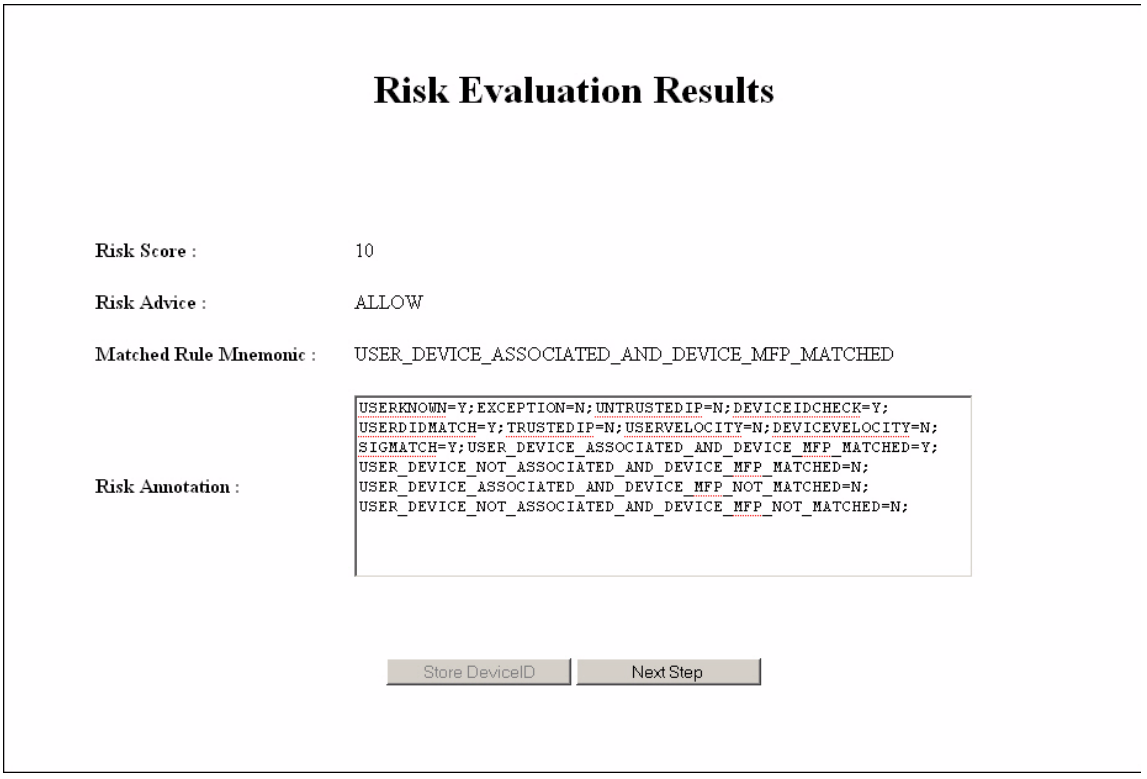
If you now select `SUCCESS` in the **Result of Secondary Authentication** field and click **Post Evaluate**, you will see the Final Risk Advice as **ALLOW**, as shown in Figure G-13.

Figure G-13 Risk Post Evaluation Result: After User was Created



After you see the **ALLOW** advice for a user, then for next risk evaluation result you will always see **ALLOW** (Figure G-14), unless you changed any on the user IP or Device ID information.

Figure G-14 Risk Post Evaluation Result: After User was Created



API Workflow and Reference

On the Create User page of Sample Application, when you click the **Create User** button:

1. The `ArRFInitHandler.class` of the `com.arcot.riskfort.sampleapp.initialize` package is invoked.

   After the initialization is complete, RiskFort is ready to serve the requests.

2. `ArRFCreateUser.jsp` calls the `ArRFCreateUserServlet.class` servlet. This servlet, in turn, calls the `ArRFUserIssuanceHelper.class` helper class in the `com.arcot.riskfort.sampleapp.helpers` package.

Important: You must ensure that `AuthProvisionFactory` has been initialized *before* you call the `createUser()` function.

As shown in `ArRFUserIssuanceHelper.class`, the most important function that you need to call from your application's servlet, .jsp, or any other calling file is the `createUser()` method.

The class and method required in the preceding workflow are described in Table G-2.

Table G-2. Main Class and Method Required for Creating Users Through Sample Application

API	Description
Class: `ArRFUserIssuanceHelper`	The helper class that contains the method required to create the specified user.
Method: `createUser()`	The method that creates the specified user.

Appendix H
Glossary

Aggregator	Third-party vendors who provides account aggregation services by collating user information across multiple enterprises.
Add-On Rule	Additional Risk Evaluation rule that ships with RiskFort.
Additional Input	Additional element pertaining to a transaction that is used by Add-On Rules for risk evaluation.
Authentication	A process by which an entity proves that it is who it claims to be.
Authentication Token	A token is an object that an authorized user of computer services is given to aid in authentication.
Callout	Custom program executing externally (outside RiskFort context).
Certificate	See Digital Certificate.
Credential	A proof of user identity. Digital credentials may be stored on hardware such as smart-cards or USB tokens or on the server. They are verified during authentication.
Customer Support Representatives (CSR)	Administrators responsible for the day-to-day operations related to users of the security system. For example, Administrators can assist users with enrollment, resetting users passwords, and generating enrollment reports.
Device Velocity	Number of transactions from the same device within a specified time.
Digital Certificate	A digital document that vouches for the identity and key ownership of an individual, a computer system, or an organization. This authentication method is based on the public-key cryptography (PKI) method.
Encryption	The process of scrambling information in a way that disguises its meaning.
Exception User	User “known” to RiskFort and is excluded from risk assessments for a specified period of time.
Evaluation Callout	Callout that runs after all Evaluation Rules and contains custom risk evaluation logic.

Evaluation Rule	Pre-configured RiskFort logic that is applied to the incoming transaction data.
Global Administrator	An administrator responsible for setting up Customer Support Representatives (CSR) accounts and configuring the system.
Increased Authentication	The Risk Advice given by RiskFort, if the current transaction is considered unsafe by RiskFort. For example, if a user does a transaction of high amount for the first time. Under such circumstances, the user is asked to re-authenticate to the authentication server through stronger authentication method.
Master Administrator	The highest level of RiskFort administrator, whose primary responsibilities are to initialize RiskFort and create Global Administrator accounts.
Negative IP Address	IP address that has been the origin of known anonymizer proxies or fraudulent or malicious transactions in the past.
Negative Country	Country from which fraudulent or malicious transactions are known to have originated in the past.
Non-Terminating Rule	The rule alone does not determine overall Risk Score. It requires other rules for the purpose.
One-Way SSL	Client application verifies the identity of the server application (by accepting server's Digital Certificate) before the SSL session is established.
Private Key	One of a pair of keys used in PKI, which is kept secret and can be used to decrypt or encrypt data.
Public Key	One of a pair of keys used in PKI, which is distributed freely and is published as part of a certificate. It is typically used to encrypt data sent to the public key's owner, who then decrypts the data using the corresponding private key.
Public Key Infrastructure (PKI)	The standards and services that facilitate the use of public-key cryptography and certificates in a networked environment.
RiskFort	RiskFort provides a mechanism to evaluate the risk of a given transaction.
Risk Advice	An action (`ALLOW`, `ALERT`, `DENY`, `INCREASEAUTH`) suggested by RiskFort to the calling application, after evaluating the risk of an transaction.

RiskFort Native Protocol	Arcot proprietary protocol for communication between RiskFort Server, its components, WebFort Server, and Administration Console.
Risk Score	RiskFort announces a score depending on the evaluation result. The score can be a number from 0 through 100. The greater the number, the higher the risk.
Scoring Callout	Callout that runs after scoring by RiskFort's Scoring Engine and contains custom scoring logic to modify final Risk Score.
Scoring Engine	Component of RiskFort Server that collects Risk Scores from individual Evaluation Rules and processes them in the order of the scoring precedence.
Scoring Rule	Last Rule that receives execution results of all other configured rules and returns the final Risk Score and Risk Advice.
Server Management Protocol	Arcot proprietary protocol for starting and shutting down the RiskFort Server.
Secure Sockets Layer (SSL)	Protocol for managing the security of a message transmission on public networks. This protocol is predecessor of Transport Layer Security (TLS).
Terminating Rule	The rule that alone determines overall Risk Score.
Transmission Control Protocol (TCP)	Internet protocol for guaranteed transmission of data. It sends data unencrypted.
Transport Layer Security (TLS)	Protocol to secure and authenticate communications across public networks by using data encryption.
Trusted Aggregator	Aggregator "trusted" to the organization and, therefore, excluded from from future risk assessments.
Trusted IP Address	IP address that is "trusted" and, therefore, excluded from future risk evaluations.
Two-Way SSL	Both, client application and the server application verify each other's identity (by presenting respective Digital Certificate) before the SSL session is established.
UserID/Password	One of the credential issued to the user during enrollment.

User Velocity	Number of transactions from the same user within a specified time.
Velocity Check	See Device Velocity and User Velocity.
Zone Hopping	Successive transactions (from same user) separated by a distance of more than what a reasonable user-speed can achieve.

Index