RIS Software Development Kit Guide

Contents

Introduction

A Software Development Kit, also called a "devkit" or SDK, is a set of development tools that allows software engineers to create customized applications for a particular software package, software framework, hardware platform, or operating system. This allows developers to create applications specific to their business needs that will interact with the product or products developed by the SDK creators.

Kount RIS Software Development Kit

The Kount RIS Software Development Kit (SDK) includes a collection of development tools that allow you to build your own customized applications in order to submit information to the Kount Risk Inquiry System (RIS) server. The RIS system is designed to receive data from the merchant's order form for evaluation and response. Once the order-form data has been received by the RIS server, Kount evaluates the inquiry and then provides a real-time response to the risks associated with the purchase and later, provides dynamic risk updates as needed[1].

In order for you to be able to receive a risk assessment from Kount for any transaction, you must submit a variety of information about the transaction by making a RIS request using an application built with the RIS SDK. This document describes how to use the RIS SDK to build applications to submit such requests. The following section outlines how this document is organized so that you can quickly locate the specific information you require.

How this document is organized

There are many methods you can use to submit a RIS request, depending on the programming language and libraries available to you. The subsequent portions in this document are organized by programming language and provide RIS SDK settings and code samples in PHP, Perl, Java, and .NET example code that will allow you to write customized applications for making RIS requests. In each language section, data is arranged as follows:

[1] See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for more information about RIS.

- **Programming Language:** The name of the programming language used for the RIS SDK. As previously mentioned, this guide presents sections for PHP, Perl, Java, and .NET.
    - **Settings:** This section contains any of the static settings necessary for the Kount RIS SDK to work correctly.
    - **Examples:** This section contains one or more code samples written in the language being referenced.
    - **Documentation:** This section contains the information about what materials you receive when you request a Kount SDK and where to find the help documentation.
- **SDK Validation:** This section of the document explains what happens if the SDK cannot validate data passed into it and is unable to send the data to RIS.

The code used in each **Examples** section of a language is sample code *only*. The examples use contrived data and only required fields. They are not suitable to be run "as is" in production. We request that you please do not copy and paste this sample code and attempt to run it. This code is intended as a guide.

NOTE: You can also choose to write RIS requests and not use the RIS SDK to create your own applications. To learn more about writing RIS requests in PHP, Perl, Java, and .NET and view code examples and sample RIS responses, see the Risk Inquiry System (RIS) chapter in the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)**.

To request the RIS SDK, please contact Kount customer support.

Authentication with Certificates and API Keys

Kount supports authentication using either Certificates or API Keys. The SDK must be set to either use a Certificate or an API Key. Which ever option is set in the SDK will be used for authentication. If it is set to use both the API Key and a Certificate, the selection will default to the API Key. If it is set to neither, authentication will not occur.

EPTOK RIS Key and Programming Language Libraries

EPTOK (Encrypted Payment Token) is an optional RIS input key. It is used to input the 1024 character RSA encrypted credit card using a public key provided by Kount.

EPTOK is handled completely "under-the-hood" and in some circumstances, the merchant may be required to include additional libraries such as for Crypt::RSA for Perl or OpenSSL for PHP. Languages such as C# and Java will not have this requirement as the libraries are compiled in.

If the merchant wants to directly input the credit card, including additional libraries will only be required if a third-party call-out requiring raw credit card numbers is used. An example of a third-party call-out is MasterCard's Lost/Stolen.

See the sections **Third Party Callout to CardinalCommerce 3-D Secure** and **MasterCard Lost Stolen Callout** sections of the **Risk Inquiry Service (RIS) Technical Specifications Guide** for more on EPTOK.

PHP

While most of the code in the SDK is the same regardless of language, there are still language specific elements in the settings that you will need to know so that the RIS SDK will work for you. The following documents the PHP-specific settings required and provides example code.

Settings

There are several required static settings that must be added so that the Kount RIS SDK will work. The settings are configured in `src/settings.ini`.

Set the following keys to the desired values:

```
MERCHANT_ID=
URL=
LOGGER=
SIMPLE_LOG_LEVEL=
SIMPLE_LOG_FILE=
SIMPLE_LOG_PATH=
PEM_CERTIFICATE=
PEM_KEY_FILE=
PEM_PASS_PHRASE=
```

The value of each of these keys needs to be populated and saved. They will rarely change[2].

- **MERCHANT_ID:** The merchant id from Kount.
- **URL:** The sandbox testing or production URL from Kount.
- **LOGGER:** Specifies which logger to use: SIMPLE or NOP. The SIMPLE logger writes messages to a specific file. NOP, the default logger, silently discards all logging.
- **SIMPLE_LOG_LEVEL:** If **SIMPLE** logging is enabled, this lists logging levels in order of decreasing severity: **FATAL**, **ERROR**, **WARN**, **INFO**, and **DEBUG**.
- **SIMPLE_LOG_FILE:** If **SIMPLE** logging is enabled, this provides the name of the log file.
- **SIMPLE_LOG_PATH:** If **SIMPLE** logging is enabled, this provides the full path to the location of the directory containing the logging file. The directory must already have been created.
- **PEM_CERTIFICATE:** Full path name of the file containing PEM formatted certificate.
- **PEM_KEY_FILE:** Full path name of the file containing private SSL key.
- **PEM_PASS_PHRASE:** Secret passphrase required to use private SSL key.

[2]See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for information about RIS Certificate Requests and Private-Key Pairs related to `Certificate_File` and `Private_Key_Password`

Examples

Once the `settings.ini` file is in place, proceed with the following.

> **NOTE:** For a complete view of the Kount SDK class library, open the `docs/index.html` file that came in the zip file.

1. Wrap the code inside of a try-catch block.
2. Create a `Kount_Ris_Request_Inquiry` object or `Kount_Ris_Request_Update` object depending on if you are sending an initial request or updating a previous transaction.
3. Call the setters required for the inquiry or update mode (**Q** is the default for inquiries and **U** is the default for updates)[3].
4. Call `getResponse()` on the inquiry or update object. A populated `Kount.Ris.Response` object will be returned. The `Kount_Ris_Response` object has many getters for using the RIS data.

Once a successful inquiry is made, incorporate as many additional setters to the object as are available before calling getResponse(). The more data provided in the object, the more accurate the risk assessment.

The following is sample code that illustrates these steps:

```
<?php

   // RIS request --- requires curl support enabled in PHP
   // php_mbstring support enabled in your php.ini file

  include 'Kount/Ris/Request/Inquiry.php';

  // Minimal RIS inquiry example:
  try {
    $inquiry = new Kount_Ris_Request_Inquiry();

    $inquiry->setSessionId('fakesessionid');
    // Go to the API_ENDPOINT https://api.kount.net/rpc/support/payments.html to see the
    // current list of accepted payment types
    // Send a credit card payment
    $inquiry->setPayment('CARD', '4111111111111111');
    //
    // The following examples are deprecated and the above example should be preferred.
    // credit card payment $inquiry->setCardPayment('4111111111111111');
    // PayPal payment: $inquiry->setPaypalPayment("testingpaypal111111");
```

[3] See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for more information on the required data


```
  // check payment: $inquiry->setCheckPayment("123456789");
  // Google payment: $inquiry->setGooglePayment("123456789012345");
  // BillMeLater payment: $inquiry->setBillMeLaterPayment("1234567890123456");
  // no payment (for example, a trial sign up): $inquiry->setNoPayment();
  $inquiry->setTotal(500);
  $inquiry->setEmail('test@kount.com');
  $inquiry->setIpAddress('192.168.0.21');
  $inquiry->setMack('Y');
  $inquiry->setWebsite("SITE-1");
  $cart = array();
  $cart[] = new Kount_Ris_Data_CartItem("TV", "LZG-123", "32-inch LCD", 1, 129999);
  $inquiry->setCart($cart);
  $inquiry->setAuth('A');
  // Setting user defined fields example
  $inquiry->setUserDefinedField("label", "value");

  // additional optional setters...

  $inquiry->setDateOfBirth("YYYY-MM-DD");
  // setGender value can be either "M" or "F"
  $inquiry->setGender("M");

  $response = $inquiry->getResponse();
  // optional getter
  $warnings = $response->getWarnings();

  print $response;

  $score = $response->getScore();
  $auto = $response->getAuto();


} catch (Exception $e) {
  print_r($e);
  // handle exception
}

// Minimal RIS update example:
try {
  $update = new Kount_Ris_Update();
  $update->setSessionId('fakesessionid');
  $update->setTransactionId($response->getTransactionId());
  $update->setMack('Y');
  $update->setAuth('A');
  // additional optional setters...
```



```
    $response = $update->getResponse();

    print $response;

    $score = $response->getScore();
    $auto = $response->geAuto();
    // additional getters etc...
  } catch (Exception $e) {
    print $e;
    // handle exception
  }
```

Documentation

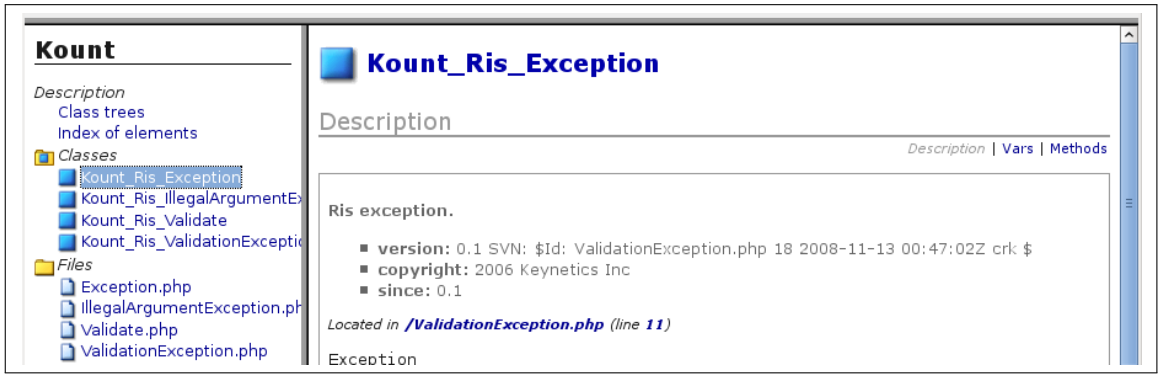
When you request the Kount RIS SDK for PHP, Kount sends you a zip file called **Sdk-Ris-Php-XXXX-YYYYMMDDTHHMM.zip**.

- **XXXX** is version number.
- **YYYY** is year.
- **MM** is the two-digit month.
- **DD** is the two-digit day.
- **T** indicates that the following value is time.
- **HH** is the 24-hour format for hour of the day.
- **MM** is the two-digit minute.

An example for version 4.3.5 is **Sdk-Ris-Php-0435-20101216T1453.zip**. When you uncompress the file, it contains the following:

- **docs:** The folder containing the documentation.
- **src:** The folder containing the source code.
- **CHANGELOG:** The text file containing Kount RIS PHP SDK changelog for the current version.
- **README:** Read Me documentation.
- **sdk-guide.pdf:** The Kount SDK Guide.

When `index.html`, contained in docs folder, opens in your web browser, you can browse the directory tree on the left and select the desired item.



Perl

While most of the code in the SDK is the same regardless of language, there are still language specific elements in the settings that you will need to know so that the RIS SDK will work for you. The following documents the Perl-specific settings required and provides example code.

Settings

There are several required static settings that must be added so that the Kount RIS Perl SDK will work. The specific settings for your environment need to be defined in the provided module `Kount::Ris::Settings`.

The settings that need to be defined are:

- **URL** is the sandbox testing or production URL from Kount.
- **MERCHANT_ID** is the merchant id from Kount.
- **HTTPS_PKCS12_FILE** is the PKCS12 certificate file exported from the browser. This RIS certificate is requested from the AWC website.
- **HTTPS_PKCS12_PASSWORD** is the password used when exporting the certificate from the browser.
- **LOGGER:** Specifies which logger to use: SIMPLE or NOP. The SIMPLE logger writes messages to a specific file. NOP, the default logger, silently discards all logging.
- **SIMPLE_LOG_LEVEL:** If **SIMPLE** logging is enabled, this lists logging levels in order of decreasing severity: **FATAL**, **ERROR**, **WARN**, **INFO**, and **DEBUG**.
- **SIMPLE_LOG_FILE:** If **SIMPLE** logging is enabled, this provides the name of the log file.
- **SIMPLE_LOG_PATH:** If **SIMPLE** logging is enabled, this provides the full path to the location of the directory containing the logging file. The directory must already have been created.

Here is an example:

```
# ris settings file
%Kount::Ris::Settings = (
  MERCHANT_ID => '999999',
  URL => 'https://example.com',
  HTTPS_PKCS12_FILE => '/path/to/certfile/cert.p12',
  HTTPS_PKCS12_PASSWORD => 'password',
  LOGGER => 'SIMPLE',
  SIMPLE_LOG_LEVEL => 'DEBUG',
  SIMPLE_LOG-PATH => '/var/log/apps',
  SIMPLE_LOG_FILE => 'Kount-Ris-Pol-SDK.log',
);
```

Examples

The general usage scenario for the Kount RIS Perl SDK is as follows:

1. Create an Inquiry or Update object for a new transaction using either `Kount::Ris::Inquiry` or `Kount::Ris::Update` modules respectively.
2. Populate the objects with as much data as is possible using the various set methods available.
3. Invoke the `getResponse()` method on the object and a `Kount::Ris::Response` object will be returned. This contains the Kount RIS repsonse data. Various get methods are provided to access this data.

The following code is an example and not meant to be run “as is”.

```
#!/usr/bin/perl

use strict;
use Kount::Ris::Inquiry;
use Kount::Ris::Update;
use Kount::Ris::Response;

my $inquiry = Kount::Ris::Inquiry->new();
$inquiry->setWebsite('DEFAULT');
$inquiry->setOrderNumber('ORDER-NUMBER');
$inquiry->setSessionId('SESSION-ID');
$inquiry->setTotal('1499');
$inquiry->setBillingAddress('1 ANY AVE', '', 'FOOLAND', 'CA', '90210');
$inquiry->setIpAddress('127.0.0.1');
$inquiry->setName("Jon Doe");
$inquiry->setEmail("jondoe@email.com");
// Go to the API_ENDPOINT https://api.kount.net/rpc/support/payments.html to see the
// current list of accepted payment types
// The setPayment(ptyp, ptok) method should be preferred to set payment information
// Send a credit card payment
$inquiry->setPayment('CARD', '4111111111111111');
$inquiry->setMack('Y');
$inquiry->addCartItem('123AD3', 'TV', 'FLATPANEL 46 INCH', '1', '249999');
$inquiry->addCartItem('893XTS', 'BLURAY', 'SAMSUMG BLURAY PLAYER', '2', '29999');
$inquiry->setUserDefinedField('MYNUMBER8', '123');
$inquiry->setUserDefinedField('CAT1', 'ALPHA TEST');
$inquiry->setAuth('A');
my $response = $inquiry->getResponse();

print $response->getVersion() . "\n";
print $response->getErrorCode() . "\n";
```

```
print $response;

my $update = Kount::Ris::Update->new();
$update->setSessionId($response->getSessionId());
$update->setTransactionId($response->getTransactionId());
$update->setMack('Y');
$update->setMode('X');
$update->setAuth('A');
my $uResponse = $update->getResponse();
print $uResponse;
```

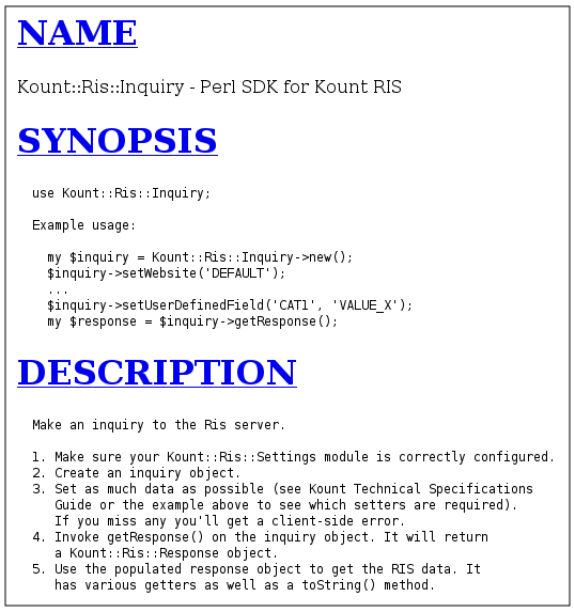
Documentation

When you request the Kount RIS SDK for Perl, Kount sends you a zip file called **Sdk-Ris-Perl-XXXX-YYYYMMDDTHHMM.zip**.

- **XXXX** is version number.
- **YYYY** is year.
- **MM** is the two-digit month.
- **DD** is the two-digit day.
- **T** indicates that the following value is time.
- **HH** is the 24-hour format for hour of the day.
- **MM** is the two-digit minute.

An example for version 4.3.5 is **Sdk-Ris-Perl-0435-20101216T1453.zip**. When you uncompress the file, it contains the following:

- **docs:** The folder containing the documentation.
- **src:** The folder containing the source code.
- **CHANGELOG:** The text file containing Kount RIS Perl SDK changelog for the current version.
- **README:** Read Me documentation.
- **sdk-guide.pdf:** The Kount SDK Guide.



NAME

Kount::Ris::Inquiry - Perl SDK for Kount RIS

SYNOPSIS

```
use Kount::Ris::Inquiry;

Example usage:

  my $inquiry = Kount::Ris::Inquiry->new();
  $inquiry->setWebsite('DEFAULT');
  ...
  $inquiry->setUserDefinedField('CAT1', 'VALUE_X');
  my $response = $inquiry->getResponse();
```

DESCRIPTION

```
Make an inquiry to the Ris server.

1. Make sure your Kount::Ris::Settings module is correctly configured.
2. Create an inquiry object.
3. Set as much data as possible (see Kount Technical Specifications
   Guide or the example above to see which setters are required).
   If you miss any you'll get a client-side error.
4. Invoke getResponse() on the inquiry object. It will return
   a Kount::Ris::Response object.
5. Use the populated response object to get the RIS data. It
   has various getters as well as a toString() method.
```

Java

While most of the code in the SDK is the same regardless of language, there are still language specific elements in the settings that you will need to know so that the RIS SDK will work for you. The following documents the Java-specific settings required and provides example code.

Settings

All of the settings for Java, such as the **Merchant ID**, **Certificate File**, and **Private Key passphrase** are hard coded. No properties file is used. See the sample code in the **Examples** section for details.

NOTE: There is no timeout set in the Java SDK. The only way to set a timeout value would be to instantiate an instance of the Transport object to override the default (which is not set to a value).

The following coding would be a possible way to override that object.

```
KountHttpTransport transport = new KountHttpTransport(password, risUrl, stream);
transport.setReadTimeout(Integer.parseInt(readTimeout));
transport.setConnectTimeout(Integer.parseInt(connectTimeout));
this.client = new KountRisClient(password, risUrl, stream); this.client.setTransport(transport);
```

Examples

1. Create an Inquiry object for a new transaction or an Update object if updating a previous transaction.
2. Depending on the mode (**Q** is default for inquiries, **U** is default for updates), there are certain required setters that must be called[4].
3. Call `getResponse()` on the inquiry or update object. It will return a populated Response object. The Response object has many getters for using the RIS data.

Once a successful inquiry is made, incorporate as many additional setters to the object as are available before calling `getResponse()`. The more data provided in the object, the more accurate the risk assessment.

The following is sample code that illustrates these steps:

```
Inquiry q = new Inquiry();
q.setMerchantId(999999);
q.setSessionId("12345678910");
q.setCurrency(CurrencyType.USD);
```

[4]See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for more information on the required data


```
q.setTotal(500);
q.setAuthorizationStatus(AuthorizationStatus.APPROVED);
q.setEmail("test@email.com");
// Go to the API_ENDPOINT https://api.kount.net/rpc/support/payments.html to see the
// current list of accepted payment types
// Send a credit card payment
q.setPayment("CARD", "4111111111111111");
q.setIpAddress("192.168.0.5");
q.setMerchantAcknowledgment(MerchantAcknowledgment.YES);
q.setWebsite("DEFAULT");

CartItem item0 = new CartItem("SURROUND SOUND", "HTP-2920",
        "Pioneer High Power 5.1 Surround Sound System", 1, 49999);
CartItem item1 = new CartItem("BLURAY PLAYER", "BDP-S500",
        "Sony 1080p Blu-Ray Disc Player", 1, 69999);
Collection<CartItem> cart = new Vector<CartItem>();
cart.add(item0);
cart.add(item1);
q.setCart(cart);

// Invoke any additional setters eg, User defined fields
KountRisClient client = new KountRisClient("abc123",
        "https://risk.test.kount.net", "kount_999999_cert.p12");

try {
    Response r = client.process(q);
    System.out.println(r);
    this.transId = r.getTransactionId();
} catch (RisException re) {
    re.printStackTrace();
}

// Minimal RIS update example
Update u = new Update();
u.setSessionId("fakesessionid");
u.setTransactionId(this.transId);
u.setMerchantAcknowledgment(MerchantAcknowledgment.NO);
u.setAuthorizationStatus(AuthorizationStatus.APPROVED);
try {
    Response r = client.process(u);
    System.out.println(r);
} catch (RisException re) {
    re.printStackTrace();
}
```

Documentation

When you request the Kount RIS SDK for Java, Kount sends you a zip file called **Sdk-Ris-Java-XXXX-YYYYMMDDTHHMM.zip**.

- **XXXX** is version number.
- **YYYY** is year.
- **MM** is the two-digit month.
- **DD** is the two-digit day.
- **T** indicates that the following value is time.
- **HH** is the 24-hour format for hour of the day.
- **MM** is the two-digit minute.

An example using version 4.3.5 is **Sdk-Ris-Java-0435-20101216T1453.zip**.

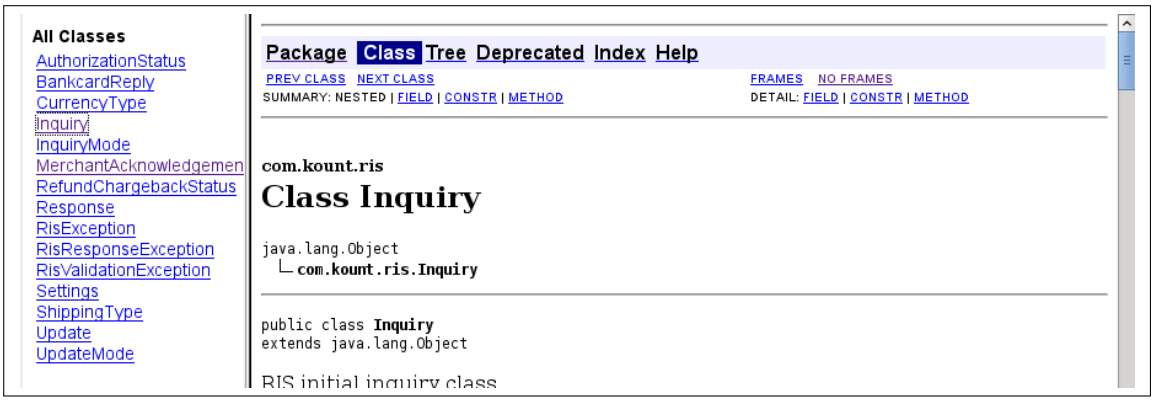
The **Java SDK** is now bundled with Apache commons logging and log4j. See:

http://commons.apache.org/logging
http://logging.apache.org/log4j

When you uncompress the file, it contains the following:

- **docs:** The folder containing the documentation.
- **lib:** The folder containing the two following artifacts:
  - **Kount-Ris.jar**
  - **logging:** A directory containing:
    - log4j.jar
    - Commons logging jar files
- **src:** The folder containing the source code.
- **CHANGELOG:** The text file containing Kount RIS Java SDK changelog for the current version.
- **README:** Read Me documentation.
- **sdk-guide.pdf:** The Kount SDK Guide.

When `index.html` opens in your web browser, you can browse the directory tree on the left and select the desired item.



.NET

While most of the code in the SDK is the same regardless of language, there are still language specific elements in the settings that you will need to know so that the RIS SDK will work for you. The following documents the .NET-specific settings required and provides example code.

IMPORTANT: You are required to create an external reference to the `KountRisSdk.dll`, most likely using **Visual Studio**, prior to following the example code provided here, in order for the RIS SDK to operate correctly. See the **Documentation** section later in this document for details.

Settings

There are several required static settings that must be added so that the Kount RIS SDK will work. These settings must be entered in a .NET generated XML file called `Web.config`, located in the directory where the application resides.

Add the following to the `appSettings` section of the Web.config file:

```
<appSettings>
    <add key="Ris.MerchantId" value="999999" />
    <add key="Ris.Url" value="https://risk.test.kount.net" />
    <add key="Ris.API.Key" value="eyJ0eXAiOiJKV.." />
    <add key="Ris.CertificateFile" value="certificate.pfx" />
    <add key="Ris.PrivateKeyPassword" value="abc123" />

    <!--
    Specify the logger to use. The default loggers supplied with the Kount RIS
    SDK are NOP (a logger that silently discards all logging), and SIMPLE (a
    simple logger that writes messages to a specified file)
    -->
    <add key="LOG.LOGGER" value="SIMPLE" />

    <!--
    Logging level for SimpleLogger if it is enabled.
    Acceptable logging levels in order of decreasing severity are FATAL,
    ERROR, WARN, INFO, and DEBUG.
    -->
    <add key="LOG.SIMPLE.LEVEL" value="DEBUG" />

    <!--
    Specify the file name where the SimpleLogger will log messages to.
    -->
    <add key="LOG.SIMPLE.FILE" value="Kount-Ris-DotNet-SDK.log" />
```



```
    <!--
    SimpleLogger log path. This is the directory where the log file will be
    located. This directory must already exist.
    -->
    <add key="LOG.SIMPLE.PATH" value="C:\Logs" />
</appSettings>
```

The value of each of these keys needs to be populated and saved. They will rarely change[5].

- **Ris.MerchantId:** the merchant id from Kount.
- **Ris.API.Key:** API key for authentication, perferred over using Certificates.
- **Ris.Url:** the sandbox testing or production URL from Kount.
- **Ris.CertificateFile:** the pk12 or pfx certificate file exported from the browser. This RIS certificate is requested from the AWC website.
- **Ris.PrivateKeyPassword:** The password used when exporting the certificate from the browser.
- **LOG.LOGGER:** Specifies which logger to use: SIMPLE or NOP. The SIMPLE logger writes messages to a specific file. NOP, the default logger, silently discards all logging.
- **LOG.SIMPLE.LEVEL:** If **SIMPLE** logging is enabled, this lists logging levels in order of decreasing severity: **FATAL**, **ERROR**, **WARN**, **INFO**, and **DEBUG**.
- **LOG.SIMPLE.FILE:** If **SIMPLE** logging is enabled, this provides the name of the log file.
- **LOG.SIMPLE.PATH:** If **SIMPLE** logging is enabled, this provides the full path to the location of the directory containing the logging file. The directory must already have been created.

Examples

Once the `Web.config` settings are in place and `KountRisSdk.dll` is referenced, proceed with the following:

1. Wrap the code inside of a try-catch block.
2. Create a `Kount.Ris.Inquiry` object or `Kount.Ris.Update` object depending on if you are sending an initial request or updating a previous transaction.
3. Call the setters required for the inquiry or update mode (**Q** is the default for inquiries and **U** is the default for updates)[6].

[5] See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for information about RIS Certificate Requests and Private-Key Pairs related to `Ris.CertificateFile` and `Ris.PrivateKeyPassword`

[6] See the **Kount Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** for more information on the required data

4. Call `getResponse()` on the inquiry or update object. A populated `Kount.Ris.Response` object will be returned. The `Kount.Ris.Response` object has many getters for using the RIS data.

Once a successful inquiry is made, incorporate as many additional setters to the object as are available before calling `getResponse()`. The more data provided in the object, the more accurate the risk assessment.

NOTE: The configuration setting to use an API Key rather than a certificate for authentication is:

```
Ris.API.Key
```

The following is sample code that illustrates these steps:

```
 // Minimal RIS inquiry example:
try
{
    Kount.Ris.Inquiry inquiry = new Kount.Ris.Inquiry();
    inquiry.SetSessionId("fakesessionid");
    // Go to the API_ENDPOINT https://api.kount.net/rpc/support/payments.html to see the
    // current list of accepted payment types
    // Send a credit card payment
    inquiry.SetPayment("CARD", "4111111111111111");
    //
    // The following examples are deprecated and the above example should be preferred.
    // credit card payment: inquiry.SetCardPayment("4111111111111111");
    // PayPal payment: inquiry.setPaypalPayment("testingpaypal111111");
    // check payment: inquiry.setCheckPayment("123456789");
    // Google payment: inquiry.setGooglePayment("123456789012345");
    // BillMeLater payment: inquiry.setBillMeLaterPayment("1234567890123456");
    // no payment (for example, a trial sign up): inquiry.setNoPayment();
    inquiry.SetTotal(1400);
    inquiry.SetEmail("test@kount.com");
    inquiry.SetIpAddress("192.168.0.21");
    inquiry.SetMack('Y');
    inquiry.SetWebsite("SITE-1");
    ArrayList cart = new ArrayList();
    cart.Add(new CartItem("TV", "LZG-123", "32-inch LCD", 1, 129999));
    inquiry.SetCart(cart);
    inquiry.SetAuth('A');
    // Setting user defined fields example
    inquiry.SetUserDefinedField("label", "value");

    // additional optional setters...

    inquiry.SetDateOfBirth("YYYY-MM-DD");
```

```
    // setGender value can be either "M" or "F"
    inquiry.SetGender(M);

    char genderChar = 'M';
    inquiry.SetGender(genderChar);

    Kount.Ris.Response response = inquiry.GetResponse();
    // optional getter
    Hashtable warnings = Response.GetWarnings();


    String score = Response.GetScore();
    String auto = Response.GetAuto();
    // etc...
}
catch (Exception e)
{
    // handle exception
}


// Minimal RIS update example:
try
{
    Kount.Ris.Update update = new Kount.Ris.Update();
    update.SetSessionId("fakesessionid");
    update.SetTransactionId(response.getTransactionId()); //
    update.SetMack('Y');
    update.SetAuth('A');
    response = update.GetResponse();

    String score = response.GetScore();
    // etc...
}
catch (Exception e)
{
    // handle exception
}
```

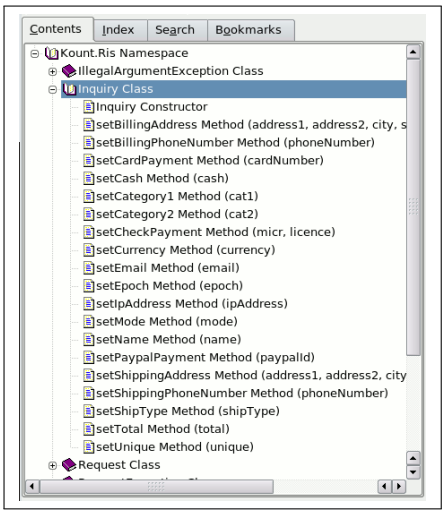

Documentation

When you request the Kount RIS SDK for .NET, Kount sends you a zip file called **Sdk-Ris-Dotnet-XXXX-YYYYMMDDTHHMM.zip**.

- **XXXX** is version number.
- **YYYY** is year.
- **MM** is the two-digit month.
- **DD** is the two-digit day.
- **T** indicates that the following value is time.
- **HH** is the 24-hour format for hour of the day.
- **MM** is the two-digit minute.

An example for version 4.3.5 is **Sdk-Ris-Dotnet-0435-20101216T1453.zip**. When you uncompress the file, it contains the following:

- **docs:** The folder containing KountRisSdk.chm
- **lib:** The folder containing KountRisSdk.dll
- **CHANGELOG.txt:** The text file containing Kount RIS .NET SDK changelog for the current version.
- **README.txt:** Read Me documentation.
- **sdk-guide.pdf:** The Kount SDK Guide.



The CHM file has been generated from the code and is the reference to the class libraries and methods. You can open the CHM file on any Windows computer by double-clicking the file, or on a Linux platform, by using a reader such as kchmviewer.

You will specifically need to consult the following Class sections of the CHM file for any settings you want to add besides the required settings listed previously in **Examples**:

- Inquiry Class
- Update Class

Each of these portions of the CHM contain the necessary information you will need to add optional settings. Open the required class book to review information on the available setters.

NOTE: If you are attempting to view the .chm file on Windows Vista or later, and the contents of the file are not visible, you may need to unblock the file. To unblock the file, right-click the .chm file, select **Properties**, and then click the **Unblock** button near the bottom of the Properties window.

SDK Validation

When data is passed into the SDK from the merchant, the SDK attempts to validate the data fields before the request is posted to RIS. The RIS post only occurs if all the data fields are validated. If the SDK cannot validate all of the fields, the RIS request is not sent. Instead, a validation exception occurs indicating the error encountered but without providing a RIS error code, since the SDK is unaware of RIS error codes.

A RIS error code is only returned if the SDK posted an invalid field that it can not validate. The SDK is only capable of validating a subset of RIS requests fields as defined in the Kount Technical Specifications Guide including those in the following list:

VERS	S2CC
MODE	S2PN
MERC	PTYP
SESS	LAST4
ORDR	DRIV
CURR	UNIQ
TOTL	EPOC
CASH	IPAD
EMAL	UAGT
GENDER	CAT1
DOB	CAT2
NAME	SHTP
B2A1	MACK
B2A2	AUTH
BPREMISE	AVSZ
BSTREET	AVST
B2CI	CVVR
B2ST	TRAN
B2CC	RFCB
B2PN	PROD_TYPE
S2NM	PROD_ITEM
S2EM	PROD_DESC
S2A1	PROD_QUANT
S2A2	PROD_PRICE
SPREMISE	B2PC
SSTREET	S2PC
S2CI	SITE
S2ST	ANID

Any bad input field caught by SDK validation will not contain a RIS error code since a RIS post does not occur.

SDK Validation Examples

Each SDK deployment zip file package contains a file called `validate.xml`. This file contains the data that the SDK uses to validate the RIS input from the merchant. The following are example XML nodes from the file with explanations:

```
<param name="CURR">
  <required>
    <mode>Q</mode>
    <mode>P</mode>
  </required>
  <reg_ex>^[A-Z]{3}$</reg_ex>
</param>
```

Explanation: When validating the RIS request field **CURR** (currency), RIS modes Q and P are required and its value must be 3 capitalized letters. The value is validated against a regular expression.

```
<param name="ORDR">
  <max_length>32</max_length>
</param>
```

Explanation: When validating the RIS request field **ORDR** (order), the maximum length is 32 characters. Note the **ORDR** field is not required.

```
<param name="SESS">
  <required />
  <max_length>32</max_length>
</param>
```

Explanation: When validating the RIS request field **SESS** (session), the maximum length is 32 characters and the field is required.

Appendix A: Kount Central SDK

To utilize the Kount Central portion of the SDK, the processor must already be enrolled. Please contact your Merchant Services representative for information about enrolling in this service.

Using Kount Central

Review the **Kount Central Implementation Guide** and the **Kount Central Technical Documentation** to familiarize yourself with this product.

Follow the steps previously outlined in this guide to setup a standard RIS request for your chosen language SDK, and then make the following additions to allow you to use Kount Central RIS modes before submitting the request.

- Set the Kount Central Customer ID
- Set the RIS mode (J or W are Kount Central modes)

Submit the request to have your Kount Central rules and threshold evaluated. To access Kount Central evaluation data please refer to your SDK's specific Kount Central accessor methods.