



# RPC

## Implementation Guide



## ***Looking for more information?***

### **Radical Blue Gaming, Inc.**

At the RadBlue forum you can find the latest release information, report program issues, get your questions answered, and submit suggestions for improving our products. Simply log on to:

<http://radblue.mywowbb.com/forum8/2.html>

+1.775.329.0990

[sales@radblue.com](mailto:sales@radblue.com)

<http://www.radblue.com>

### **Gaming Standards Association**

If you need a copy of the Gaming Standards Association's latest S2S protocol documents, or to find out more about the GSA and the work being done by over 70 companies in the areas of protocol standardization for the gaming industry, we encourage you to discover more about the organization.

<http://www.gamingstandards.com>

[sec@gamingstandards.com](mailto:sec@gamingstandards.com)

**Copyright 2008 Radical Blue Gaming, Inc. All rights reserved.**

All trademarks used within this document are the property of their respective owners.

No part of this work may be reproduced in whole or in part, in any manner, without the prior written permission of Radical Blue Gaming, Inc.



## Revision

---

Date	Revision	Description
07 APR 2008	1.2	Added the following fields to <b>voucherTicketData</b> : <b>propertyName</b> , <b>propertyAddr1</b> , <b>propertyAddr2</b> , <b>titleCash</b> , <b>titleLargeWin</b> , <b>titleNonCash</b> , <b>titleNonPromo</b> and <b>titlePromo</b> .
20 MAR 2008	1.2	Added <b>disconnectFromVoucherSystem</b> call information.
14 FEB 2008	1.1	Added <b>connectToVoucherService</b> and <b>cancelRedeem</b> call information.
30 JAN 2008	1.0	New document.





# Contents

---

<b>Revision</b> .....	<b>iii</b>
<b>Chapter 1: Introducing RPC</b> .....	<b>1-1</b>
About the RPC .....	1 - 1
The RPC Instance .....	1 - 2
RPC Simulators .....	1 - 3
Voucher Service API Calls .....	1 - 4
About the Derby Database .....	1 - 5
Supported Voucher Systems .....	1 - 5
RPC Server Requirements .....	1 - 5
<b>Chapter 2: RPC API Flow</b> .....	<b>2-1</b>
<b>Chapter 3: Developing an RPC Interface</b> .....	<b>3-1</b>
Getting Started with RPC Interface Development .....	3 - 1
RPC Development Roadmap .....	3 - 1
Install the RPC API and Simulators .....	3 - 2
Modify Configuration File(s) .....	3 - 2
Purpose .....	3 - 2
Procedure .....	3 - 3
Set Up Authentication and Encryption .....	3 - 4
Start the RPC Server .....	3 - 4
Procedure .....	3 - 4
Running Multiple Instances of RPC .....	3 - 5
About the Kiosk Client Simulator .....	3 - 6
Purpose .....	3 - 6
Using the Kiosk Client Simulator .....	3 - 7

Initialize Connection . . . . .	3 - 8
Disconnect RPC Connection from Voucher System . . . . .	3 - 8
Verify RPC . . . . .	3 - 9
Verify Voucher System . . . . .	3 - 9
RPC Checksum . . . . .	3 - 10
Register Kiosk . . . . .	3 - 10
Create Voucher . . . . .	3 - 11
Confirm Voucher . . . . .	3 - 12
Redeem Voucher . . . . .	3 - 13
Commit Voucher . . . . .	3 - 14
Cancel Redeem Voucher . . . . .	3 - 16
Void Voucher . . . . .	3 - 16
Generate Report . . . . .	3 - 17
Unregister Kiosk . . . . .	3 - 18
About the Mock Host Simulator . . . . .	3 - 20
Using the Logging Window . . . . .	3 - 20
Message Format . . . . .	3 - 20

## **Chapter 4: Voucher Service API . . . . . 4-1**

connectToVoucherService . . . . .	4 - 1
Purpose . . . . .	4 - 1
Parameters . . . . .	4 - 1
Example Configuration Strings . . . . .	4 - 1
Return . . . . .	4 - 2
isRpcActive . . . . .	4 - 3
Purpose . . . . .	4 - 3
Parameters . . . . .	4 - 3
Return . . . . .	4 - 3
isVoucherSystemActive . . . . .	4 - 3
Purpose . . . . .	4 - 3
Parameters . . . . .	4 - 3
Return . . . . .	4 - 3
registerKiosk . . . . .	4 - 4
Purpose . . . . .	4 - 4
Parameters . . . . .	4 - 4
Return . . . . .	4 - 4
unregisterKiosk . . . . .	4 - 5
Purpose . . . . .	4 - 5
Parameters . . . . .	4 - 5
Return . . . . .	4 - 5
createVoucher . . . . .	4 - 6
Purpose . . . . .	4 - 6
Parameters . . . . .	4 - 6

Return .....	4 - 6
confirmVoucher .....	4 - 7
Purpose .....	4 - 7
Parameters .....	4 - 7
Return .....	4 - 7
voidVoucher .....	4 - 8
Purpose .....	4 - 8
Parameters .....	4 - 8
Return .....	4 - 8
redeemVoucher .....	4 - 9
Purpose .....	4 - 9
Parameters .....	4 - 9
Return .....	4 - 9
cancelRedeem .....	4 - 10
Purpose .....	4 - 10
Parameters .....	4 - 10
Return .....	4 - 10
commitVoucher .....	4 - 11
Purpose .....	4 - 11
Parameters .....	4 - 11
Return .....	4 - 11
generateReport .....	4 - 12
Purpose .....	4 - 12
Parameters .....	4 - 12
Return .....	4 - 12
verifyChecksum .....	4 - 13
Purpose .....	4 - 13
Parameters .....	4 - 13
Return .....	4 - 13
disconnectFromVoucherService .....	4 - 13
Purpose .....	4 - 13
Parameters .....	4 - 13
Return .....	4 - 13
Call Responses .....	4 - 14
voucherServiceResult .....	4 - 14
voucherTicketData .....	4 - 15
voucherReportEntry .....	4 - 16
voucherTicketRedemptionData .....	4 - 17

**Appendix A: Example Configuration Files . . . . . A-1**

RPC\_CONFIG MOCK.XML . . . . . A - 1

RPC\_CONFIG CTN.XML . . . . . A - 2

RPC\_CONFIG S2S1-2-6.XML . . . . . A - 3

RPC\_CONFIG STC9X.XML . . . . . A - 4

**Appendix B: SOAP Errors. . . . . B-1**

**Index . . . . . IX-1**

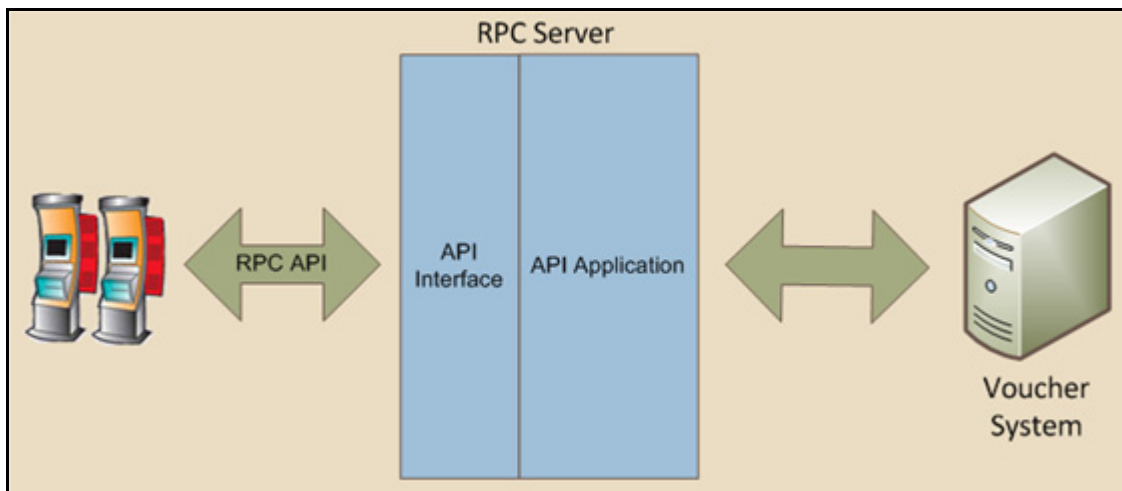


# Chapter 1

## Introducing RPC

### About the RPC

The RadBlue Protocol Converter (RPC) provides a single point of communication between kiosks, or a kiosk server, and various voucher systems. The *RPC Implementation Guide* and RPC simulators provide you with the tools you need to create an interface between your kiosks, or kiosk server, and the RPC server.



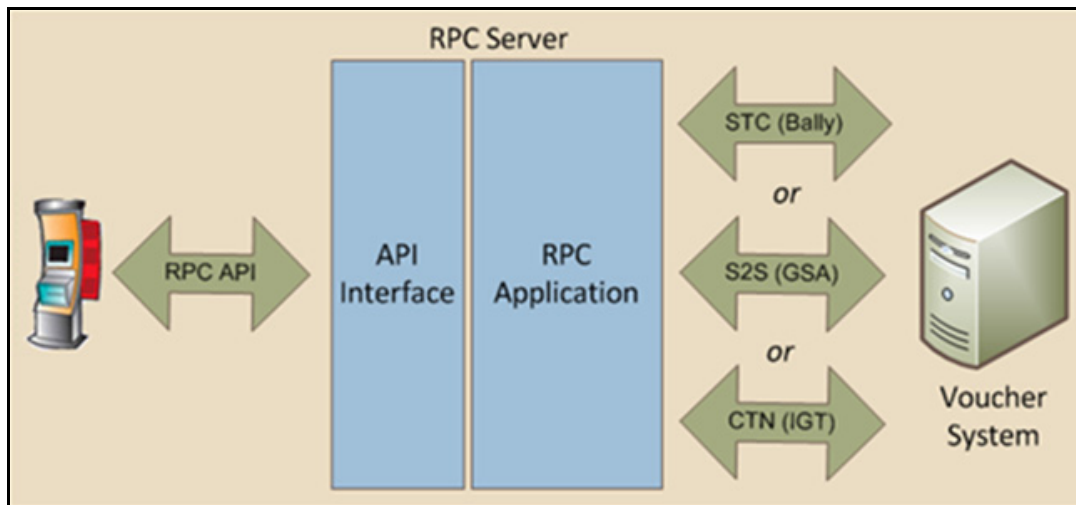
**Figure 1 - 1:** RPC Overview

Interfacing to the RPC is accomplished through a Simple Object Access Protocol (SOAP) service. The RPC Application Protocol Interface (API) is *stateless*, meaning each call is independent of the other calls.

Each kiosk has a unique identifier comprised of the property ID, kiosk ID and password. Most communication to the RPC requires these three pieces of information *at a minimum*. This information is provided by the voucher server (host).

## The RPC Instance

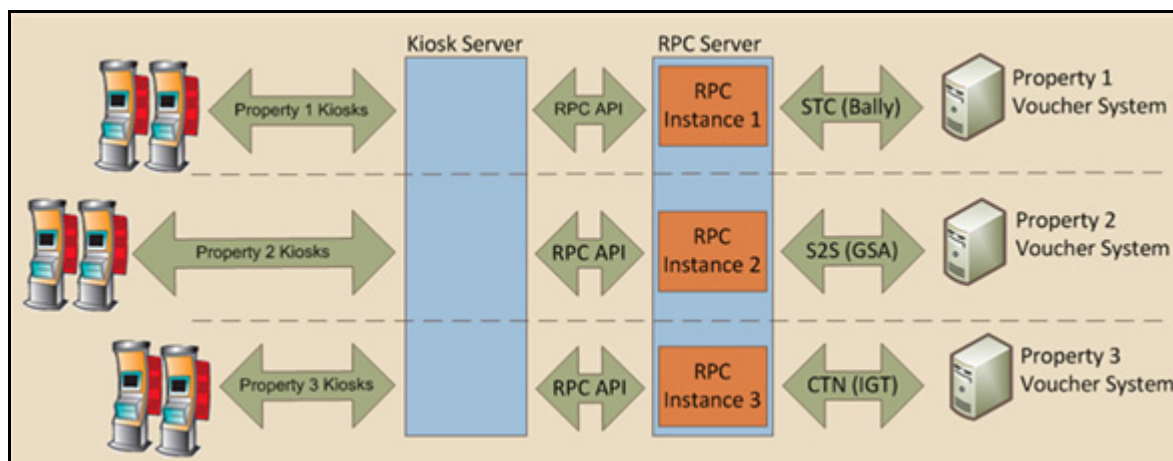
The RPC represents a software instance that can be run on any Windows or Linux server. For larger properties, a dedicated server can run a single instance of the RPC application. The RPC is configured to connect to one of three valid voucher protocols - STC, S2S and CTN - or to the mock host simulator. Figure 1-2 shows the configuration for a dedicated server.



**Figure 1 - 2:** Dedicated RPC server

Any single instance of the RPC server can only be configured to connect to a single voucher server.

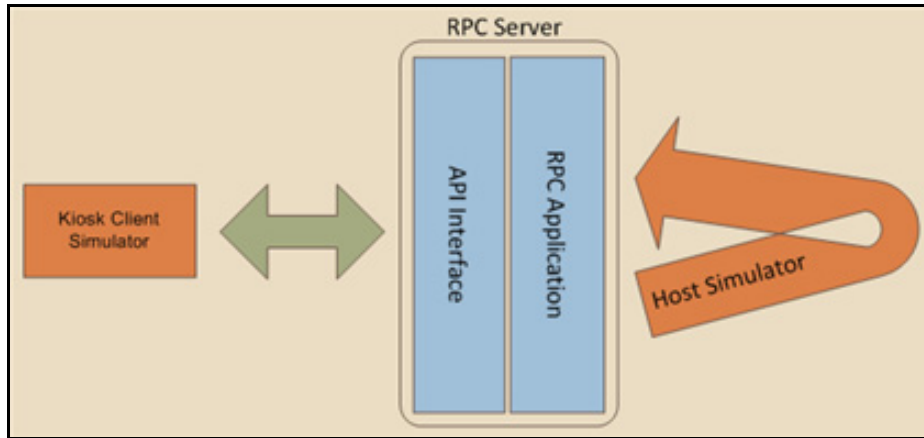
For smaller properties, multiple RPC instances can be configured to run on a single hardware server. As in the dedicated server case, each of the RPC instances communicates to a single voucher server (Figure 1-3).



**Figure 1 - 3:** Shared RPC server

## RPC Simulators

There are two simulators available to assist you with RPC interface development: the mock host simulator and the kiosk client simulator.

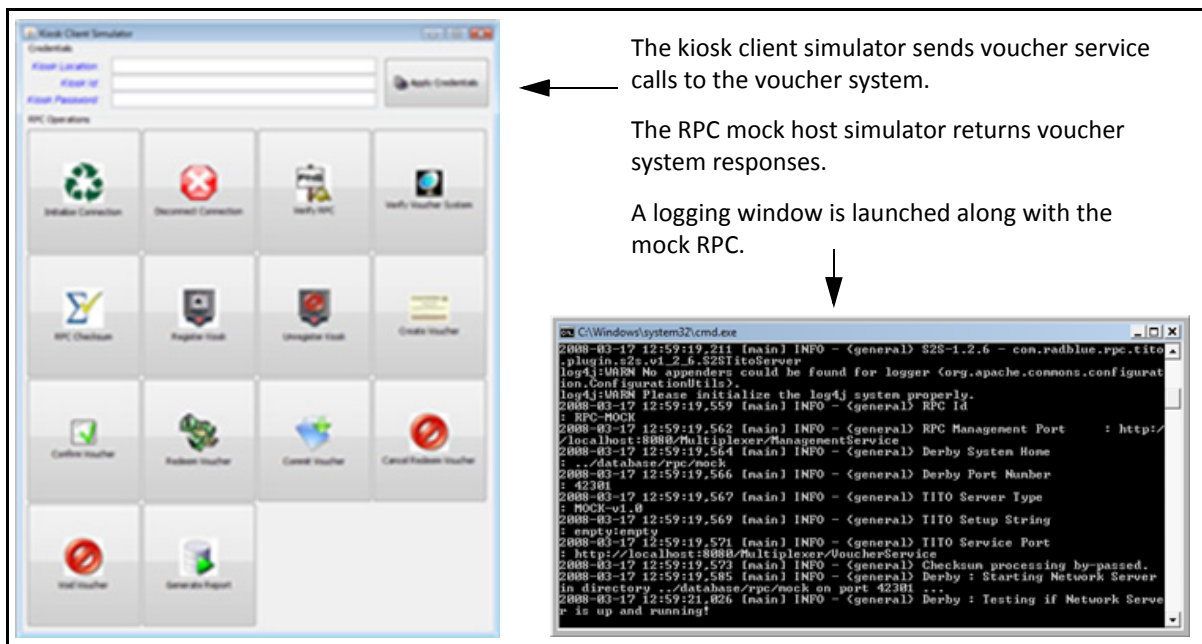


**Figure 1 - 4:** RPC simulator flow.

The mock host simulator host is used in the absence of a connection to an actual host. It simulates host commands and responses to the kiosk.

The kiosk client simulator simulates kiosk commands and responses to the host.

Both simulators can be used together to demonstrate the message flow between the kiosk and host (Figure 1-4). The mock interface allows you to view the messages traveling between the simulators.



**Figure 1 - 5:** Simulator interfaces.

## Voucher Service API Calls

The following are Voucher Service API calls:

Call	Description
connectToVoucherSystem	Used to initiate a connection between RPC and the voucher system.
isRpcActive	Used to verify whether the RPC service is running and active.
isVoucherSystemActive	Used to verify whether the configured voucher service is running and active.
registerKiosk	This call informs the RPC when a particular kiosk becomes active. Used when a kiosk starts.
createVoucher	Used to create a voucher.
unregisterKiosk	This call informs the RPC when a particular kiosk is taken offline. The RPC does not accept calls from kiosks that are not registered.
confirmVoucher	Used to confirm that a voucher was successfully printed.
redeemVoucher	Used to send a voucher redemption request to the voucher system.
cancelRedeem	Used to cancel a voucher redemption operation. This call must follow a <b>redeemVoucher</b> call.
commitVoucher	Used to report the results of the prior voucher redemption transaction. It is sent to report what happened, even if the redemption was not successful.
voidVoucher	Used to void a voucher. A voided voucher is no longer eligible for redemption. Whether a kiosk can void only vouchers that it issued, or all vouchers, depends on the voucher system functionality. RPC supports either option.
generateReport	Used to generate a report of voucher activity.
verifyChecksum	Used to verify that the RPC server.
disconnectFromVoucherService	Used to sever the connection between RPC and the voucher system.

Most API calls require that each kiosks property ID, kiosk ID and password fields be sent from the host, with the exception of **isVoucherSystemActive**, **isRpcActive** and **verifyChecksum**. *For each call, the caller must supply all required information.*

## About the Derby Database

RPC uses of the *Derby* database. Derby is a light-weight database. In most cases, the database is simply used to keep track of registered kiosks. This list is sent to the CTN or S2S servers upon startup. The registered list is not required by STC.

If specified in the [connectToVoucherService](#) API call, the RPC can also be instructed to keep track of all the voucher transactions that take place. It is important to note that if this feature is enabled, there may be an impact on performance. This is because all voucher transactions are logged in the database.

Each instance of the RPC has it's own instance of Derby. You can change the port and location of the data file by modifying the the RPC configuration files.

## Supported Voucher Systems

The current RPC version connects to the following systems:

- IGT Advantage or EZPay, using the CTN protocol
- Bally Technologies SDS, using the STC protocol
- Any system compliant with the GSA S2S 1.2.6 protocol

## RPC Server Requirements

The RPC server is a Java application, running under the Java 1.6.1-01 environment

- Processor: Intel 1.8Ghz
- Memory: 1GB
- Hard Drive: 100GB
- Operating System: Windows XP Professional
- Runtime Environment: Java 1.6.1





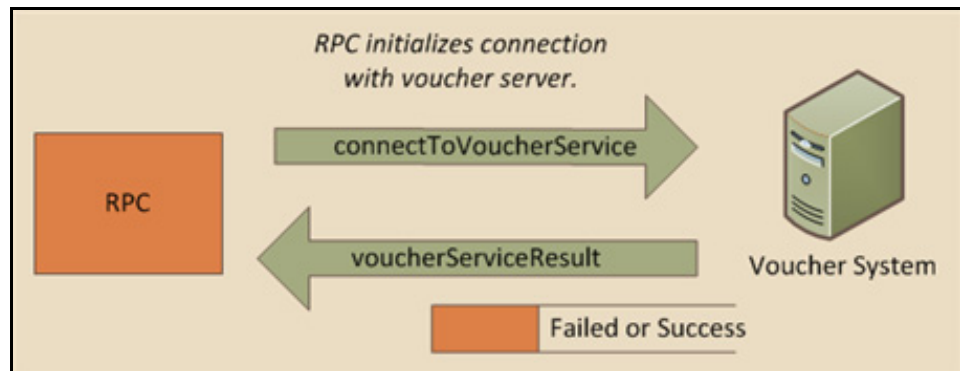
# Chapter 2

## RPC API Flow

The following describes the general flow of API calls:

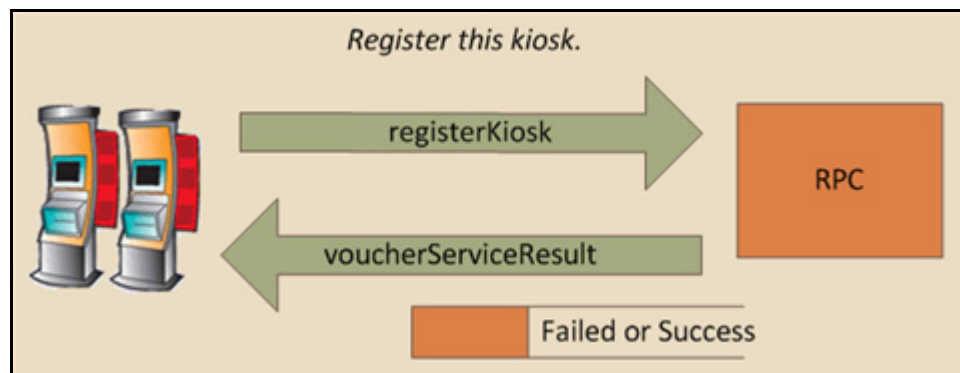
### 1. Initialize connection to the voucher system.

The **connectToVoucherSystem** function is used to start a connection with the voucher system.



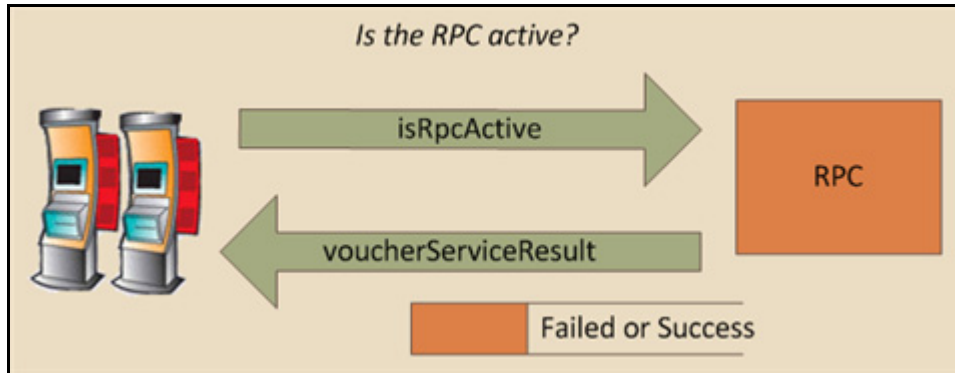
### 2. Register the kiosk.

The first thing that a kiosk should do when it starts is to initialize its connection with the voucher system through the **registerKiosk** function. This function registers the kiosk with the RPC, allowing the RPC to accept future requests from the kiosk.



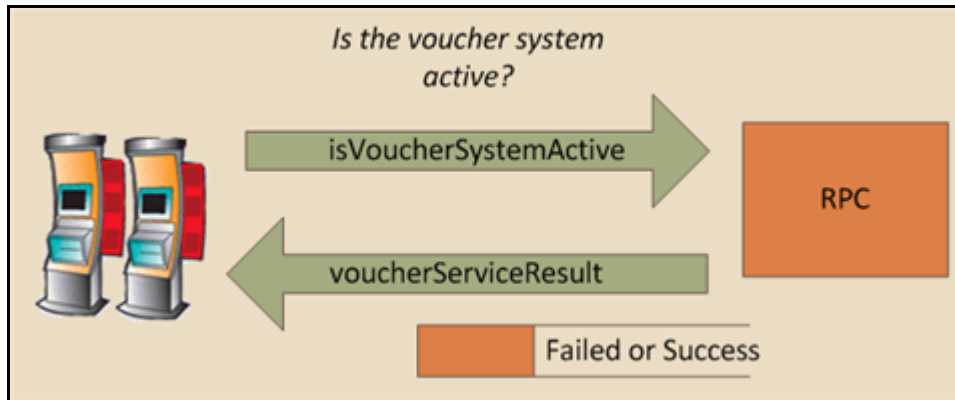
### 3. Verify that the RPC is active.

The **isRpcActive** function can be sent anytime by a kiosk. This function is used by the kiosk to test whether the RPC is running and active.



### 4. Verify that the voucher system is active.

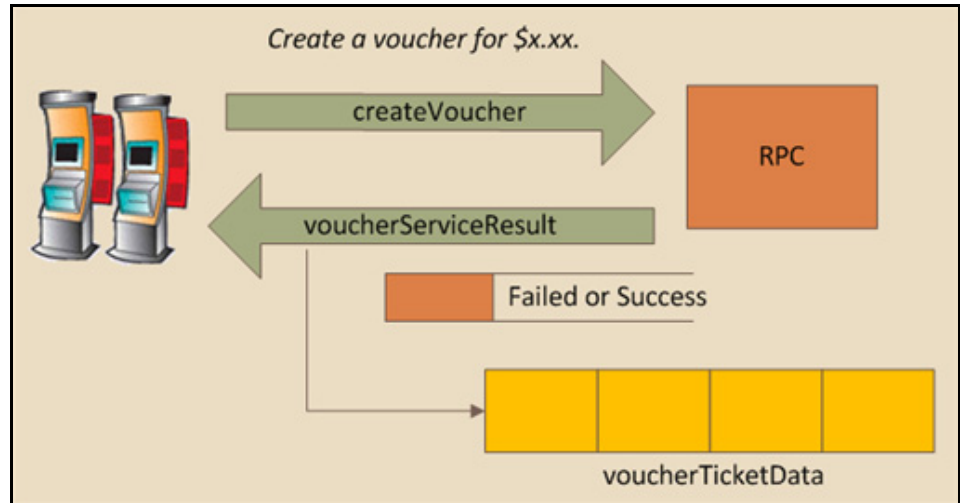
The **isVoucherSystemActive** function can be sent anytime by a kiosk. This function is used by the kiosk to test whether the voucher system is running.



## 5. Create a voucher.

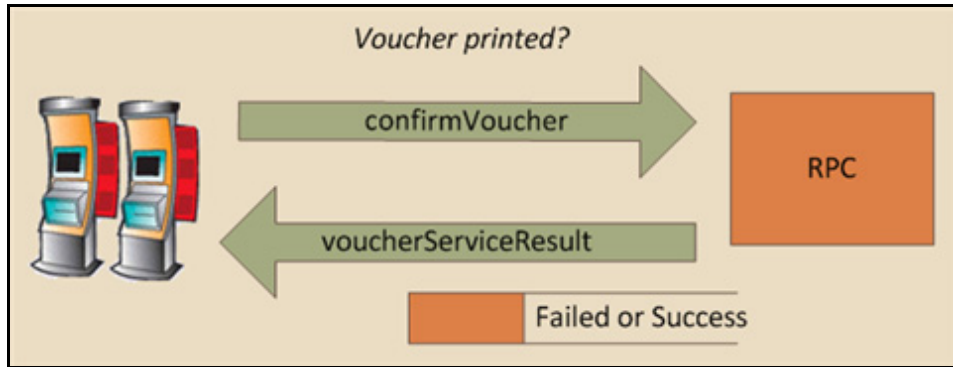
The **createVoucher** function is sent when a new voucher is to be printed. The RPC returns a **voucherServiceResult** object containing the **voucherTicketData**.

The **voucherTicketData** object contains fields to be printed on the physical voucher, including: barcode, player ID, location, voucher amount, effective date, expire date, and voucher status.

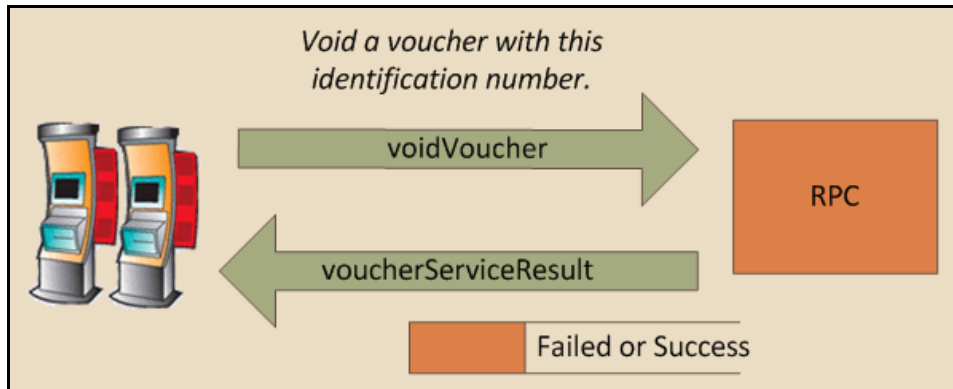


## 6. Verify that the voucher was printed successfully.

If the voucher was printed successfully, a **confirmVoucher** should be called. This informs the RPC server that the patron has received the voucher.



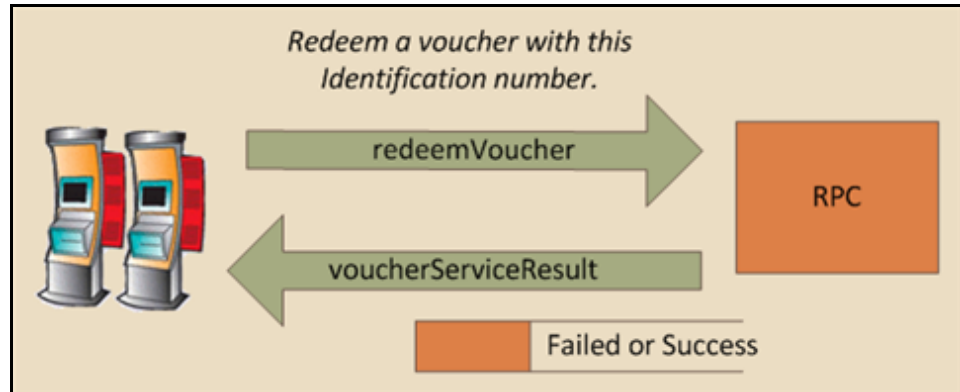
If there was an issue printing the voucher, the **voidVoucher** is sent. The RPC relays to the voucher server that there was a problem and that the voucher must be voided.



## 7. Redeem a voucher.

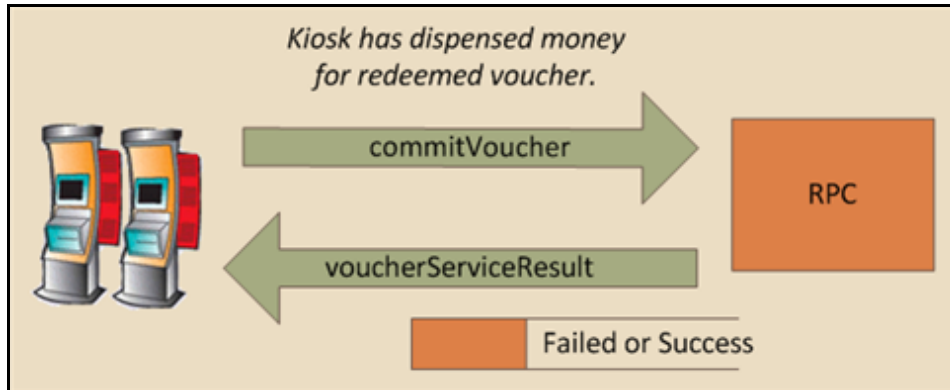
The **redeemVoucher** call is sent to request the redemption of a presented ticket. The kiosk only knows the barcode and needs to know the amount and whether the voucher is eligible for redemption.

The RPC returns a **voucherServiceResult** object with the redemption status. The response must contain all of the required voucher information (including voucher amount and fund type) *or* an error indicating that the voucher is not redeemable.

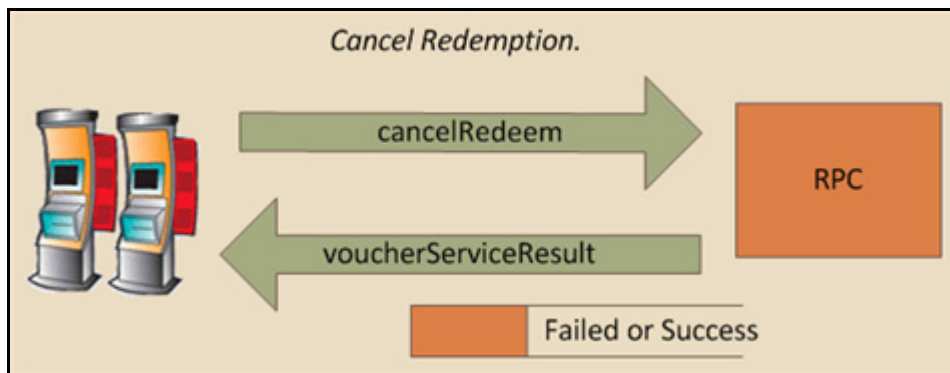


### 8. Notify host of voucher redemption transaction result

The **commitVoucher** call reports the outcome of the voucher redemption transaction. This call must be sent *after* the **redeemVoucher** call.

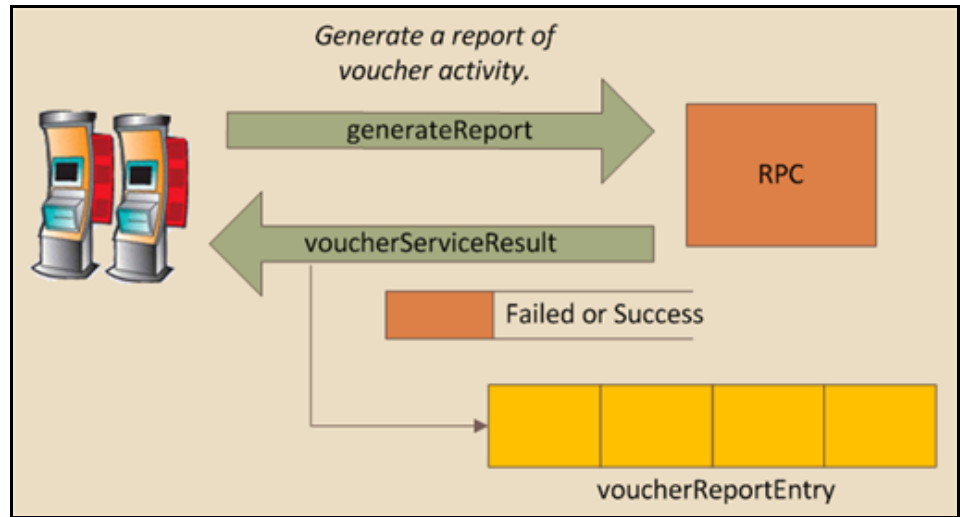


Or, cancel the voucher redemption by sending a **cancelRedeem** call. This call must be sent *after* the **redeemVoucher** call.



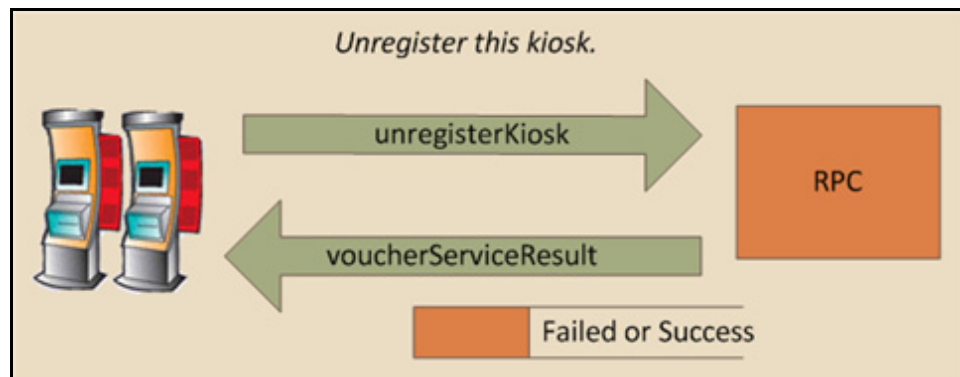
## 9. Generate a voucher report

The **generateReport** call can be sent at anytime. It provides a voucher report of all vouchers (issued, redeemed, voided) between a specified time period. The information returned in the **voucherReportEntry** response includes: barcode, transaction location, transaction date and time, voucher amount, and employee who voided the voucher (if applicable).



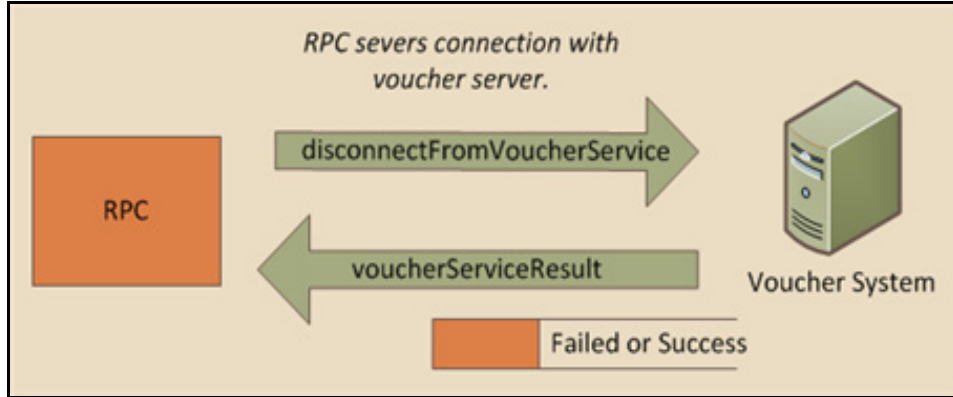
## 10. Unregister the kiosk.

The **unregisterKiosk** call notifies the voucher system that the kiosk is inactive. Once a kiosk is unregistered, the RPC no longer accepts requests from it.



### 11. Disconnect from the voucher system.

The **disconnectFromVoucherSystem** function is used to sever the connection with the voucher system.





# Chapter 3

## Developing an RPC Interface

---

### Getting Started with RPC Interface Development

This chapter provides information to help you begin implementing RPC quickly. To help you in the development process, we provide the following:

- Mock Host Simulator
- Kiosk Client Simulator
- Voucher Service API Call Format
- Sample Configuration Files
- Voucher Service API Error Codes

#### RPC Development Roadmap

1. Install the RPC API and Simulators
2. Configure the Host Protocol File(s)
3. Set Up Authentication and Encryption
4. About the Kiosk Client Simulator
5. About the Mock Host Simulator

## Install the RPC API and Simulators

RPC is distributed in **.zip** format. Simply unzip the contents into the folder of your choice. The following folders are created in the **rpc** root directory:

Folder	Description
BIN	Startup files for RPC components.
CONFIG	All configuration files.
LIB	All application <b>.jar</b> and <b>.class</b> files.
JRE	Java runtime environment.
DATABASE	Databases for RPC components.
SCHEMAS	RPC API schema and <b>.wsdl</b> files.
DOCS	RPC documentation.

## Modify Configuration File(s)

### Purpose

To connect to the host system by modifying site-specific attributes in the configuration file. The file(s) that you configure depends on the protocol(s) to which you are connecting:

- Radblue mock - **rpc\_config\_mock.xml**
- IGT CTN - **rpc\_config\_ctn.xml**
- GSA S2S - **rpc\_config\_s2s1-2-6.xml**
- Bally Technologies STC - **rpc\_config\_stc9x.xml**

To view example code for each configuration file, see *Chapter A: Example Configuration Files* on page A-1.

## Procedure

1. Navigate to **rpc > config**.
2. Right-click the file you want to modify, and select **Edit**.
3. Modify the attributes specified below, as appropriate.

Attribute	Description
<b>rpc.id</b>	RPC instance name. This field can be configured to any descriptor, as defined by the site.
<b>derby.drda.portNumber</b>	Database where the voucher data is stored. The port number should be an integer. <i>This information is provided by the host system administrator.</i>
<b>system.home</b>	Location on the database where the voucher data is stored. <i>This information is provided by the host system administrator.</i>
<b>tito.server.type</b>	Protocol used by the voucher system to which the client is connecting: <ul style="list-style-type: none"> <li>• RadBlue - <b>MOCK-v1.0</b></li> <li>• IGT - <b>CTN</b></li> <li>• GSA - <b>S2S-1.2.6</b></li> <li>• Bally Technologies - <b>STC-9.x</b></li> </ul>
<b>tito.server.setup</b>	IP address of the host system. <i>This information is provided by the host system administrator.</i>
<b>voucher.service.port</b>	Port number to which the client connects. The port number should be an integer. <i>This information is provided by the host system administrator.</i>  <b>NOTE:</b> If you are configuring the mock host simulator, enter your computer's IP address.
<b>management.service.port</b>	<i>For future use.</i>  <b>NOTE:</b> If you are configuring the mock host simulator, enter your computer's IP address.

4. Select **File**, and click **Save**.

## Set Up Authentication and Encryption

In addition to the above protocol-specific configuration files, a Certificate Authority (**cacerts**) file must be configured with protocol-specific information. The **cacerts** file contains messaging authentication and encryption information. The **cacerts** file must be configured with information provided by the system vendor.

*Contact the voucher system vendor for more information.*

## Start the RPC Server

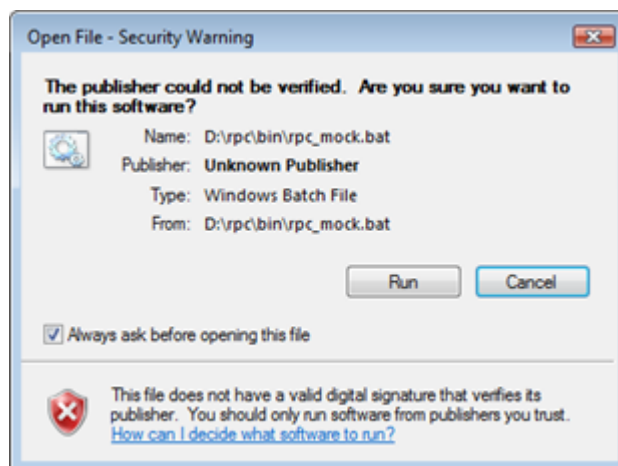
A **.bat** file starts the RPC server. There is a **.bat** file for each supported protocol, including the mock host simulator:

- Radblue host simulator - **rpc\_mock.bat**
- Bally Technologies STC protocol - **rpc\_stc.bat**
- IGT CTN protocol - **rpc\_ctn.bat**
- GSA S2S protocol - **rpc\_s2s.bat**

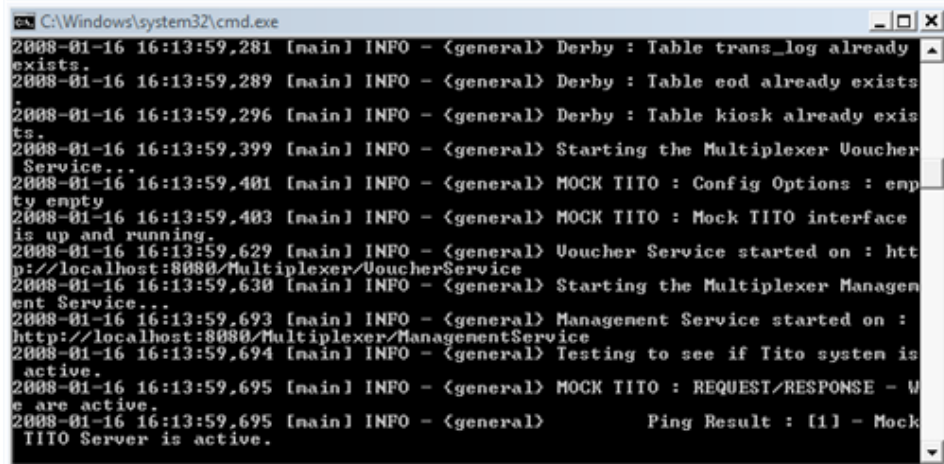
The following information applies to all **.bat** files.

### Procedure

1. Go to the **rpc** directory you installed, and double-click **bin**.
2. Double-click **rpc\_x.bat**, where *x* is the host system protocol.



### 3. Click **Run** to start the RPC.



```
C:\Windows\system32\cmd.exe
2008-01-16 16:13:59,281 [main] INFO - <general> Derby : Table trans_log already exists.
2008-01-16 16:13:59,289 [main] INFO - <general> Derby : Table eod already exists.
2008-01-16 16:13:59,296 [main] INFO - <general> Derby : Table kiosk already exists.
2008-01-16 16:13:59,399 [main] INFO - <general> Starting the Multiplexer Voucher Service...
2008-01-16 16:13:59,401 [main] INFO - <general> MOCK TITO : Config Options : empty empty
2008-01-16 16:13:59,403 [main] INFO - <general> MOCK TITO : Mock TITO interface is up and running.
2008-01-16 16:13:59,629 [main] INFO - <general> Voucher Service started on : http://localhost:8080/Multiplexer/VoucherService
2008-01-16 16:13:59,630 [main] INFO - <general> Starting the Multiplexer Management Service...
2008-01-16 16:13:59,693 [main] INFO - <general> Management Service started on : http://localhost:8080/Multiplexer/ManagementService
2008-01-16 16:13:59,694 [main] INFO - <general> Testing to see if Tito system is active.
2008-01-16 16:13:59,695 [main] INFO - <general> MOCK TITO : REQUEST/RESPONSE - We are active.
2008-01-16 16:13:59,695 [main] INFO - <general> Ping Result : [1] - Mock TITO Server is active.
```

## Running Multiple Instances of RPC

Each **.bat** file must have a corresponding configuration file. To run more than one instance using the same protocol, a different configuration file is required for each instance.

1. Save a copy of the configuration file with whatever naming convention you want
2. Save a copy of the **.bat** file with whatever naming convention you want
3. Right-click the saved **.bat** file and select **Edit**
4. Change the following line with the saved configuration file name:  
`..\jre1.6.0\bin\java ... ../config/rpc_config_mock.xml  
../config/plugin_definitions.xml`
5. Select **File > Save**, and close the file.

## About the Kiosk Client Simulator

### Purpose

The purpose of the kiosk client simulator is to see how client calls are sent to the RPC server and the responses that are returned from the host. The host simulator can also be used in the absence of a connection to an actual host.

To ensure that the kiosk client simulator works with the host simulator, the host simulator must be launched *and active* prior to launching the kiosk client simulator.

1. Go to the **rpc** directory you installed, and double-click **bin**.
2. If you are using the kiosk client simulator with the mock host simulator, follow the configuration instructions below. If you are connecting to any other protocol, proceed to step 3.

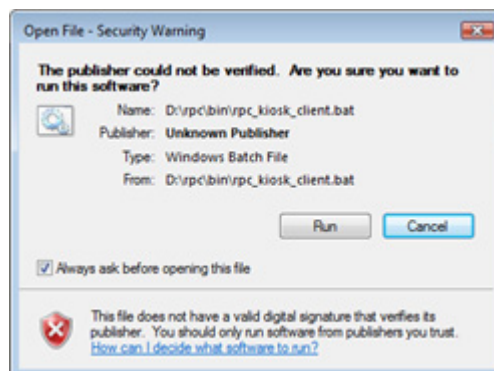
To edit the **rpc\_kiosk\_client.bat** file:

- Highlight the file, right-click, and select **Edit**.
- When prompted, click **Run**.
- At the following line, change **localhost:8080** to **xxx.xxx.xxx.xxx: 8080**, where **xxx.xxx.xxx.xxx** is your computer's IP address:

```
..\jre1.6.0\bin\java ... http://localhost:8080/Multiplexer/ VoucherService
```

- Click **Save**, and close the file.
- Proceed to step 3.

3. Double-click **rpc\_kiosk\_client.bat**

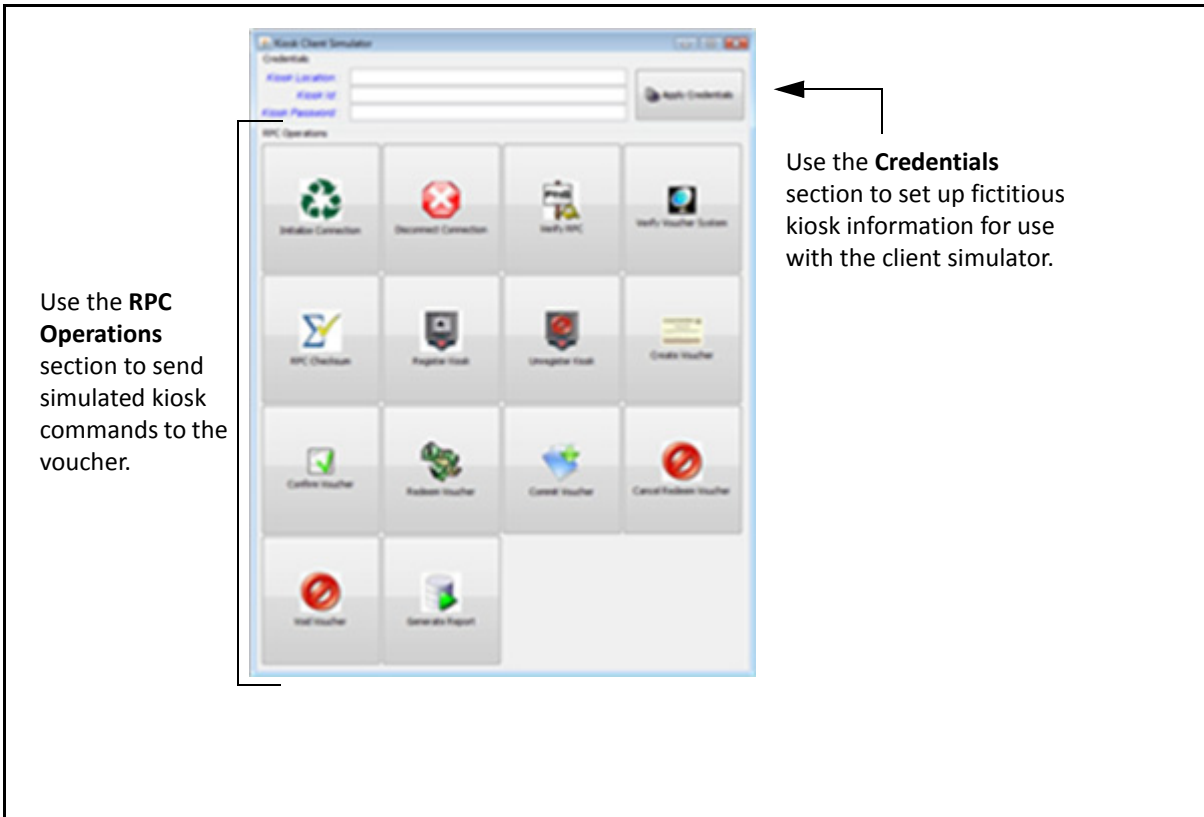


4. Click **Run**.

### Using the Kiosk Client Simulator

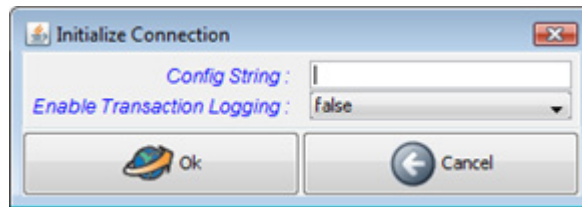
To use the kiosk client simulator, simply enter the fictitious kiosk information to be used in the simulation in the **Kiosk ID**, **Kiosk Location**, and **Kiosk Password** fields. Click **Apply Credentials** to use the entered credentials.

Note that, when you click **Apply Credentials**, the information is not sent to the host simulator. Rather, this information is used to auto-populate fields for the calls sent through the buttons under RPC Operations. There are no restrictions on the information entered into these fields.



### Initialize Connection

1. Click **Initialize Connection** to initiate the connection with the voucher server.



2. Enter the **Configuration String** to be used to initialize the connection to the voucher server.

#### Format:

<voucher server IP address>|<voucher server port>|<CTN TYPE>|<CTN ID>

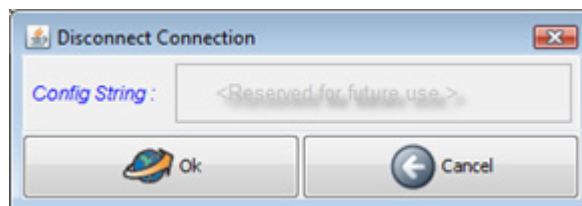
#### Example:

192.168.254.22|40666|QUIKPLAY|49160

3. In the **Enable Transaction Logging** drop-down, select whether RPC enters transactions into its local database. “True” means that transactions will be entered into the database; “False” means that transactions will not be entered into the database.
4. Click **OK** to send the **connectToVoucherService** call.

### Disconnect RPC Connection from Voucher System

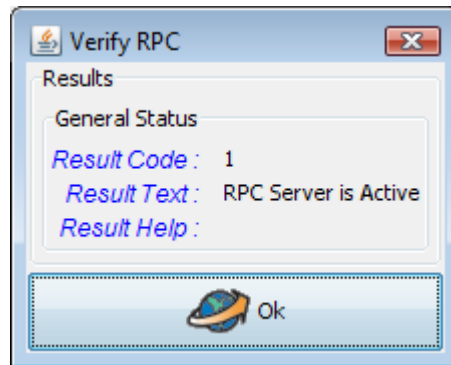
1. Click **Disconnect Connection** to sever the RPC connection with the voucher server.



2. Click **OK** to send the **disconnectFromVoucherService** call.

### Verify RPC

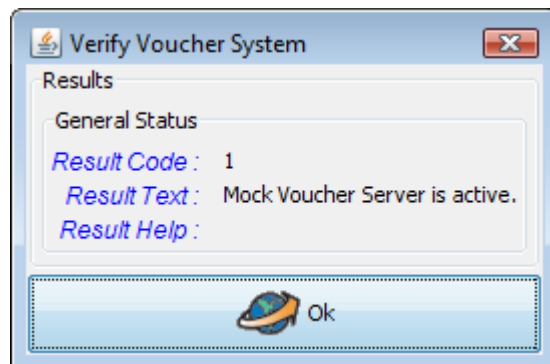
1. Click **Verify RPC** to send a **isRpcActive** call.



2. Click **OK**.

### Verify Voucher System

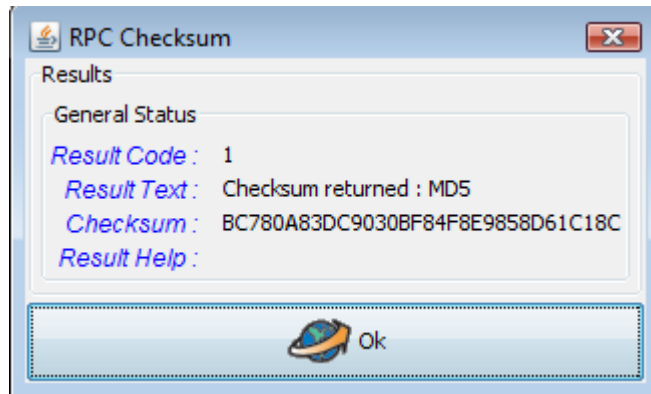
1. Click **Verify Voucher System** to send the **isVoucherSystemActive** call.



2. Click **OK**.

### RPC Checksum

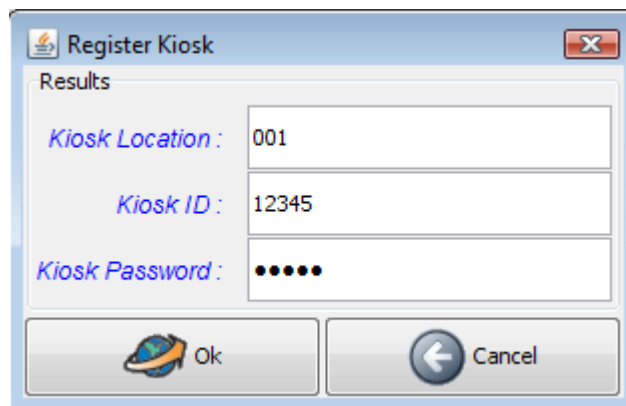
1. Click **RPC Checksum** to send the **verifyChecksum** call.



2. Click **OK**.

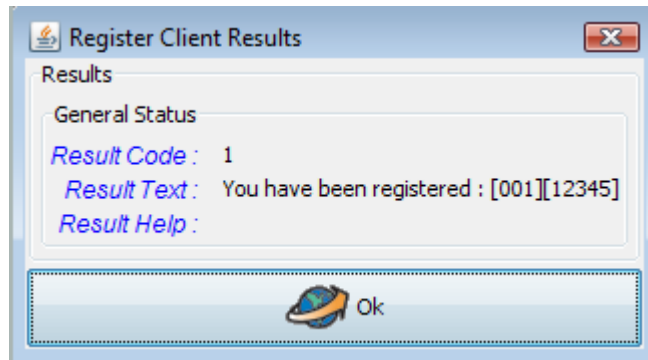
### Register Kiosk

1. Click **Register Kiosk**.



2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.

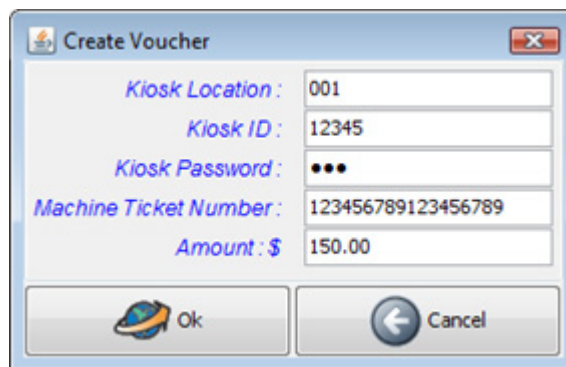
5. Click **OK** to send a **registerKiosk** call.



6. Click **OK**.

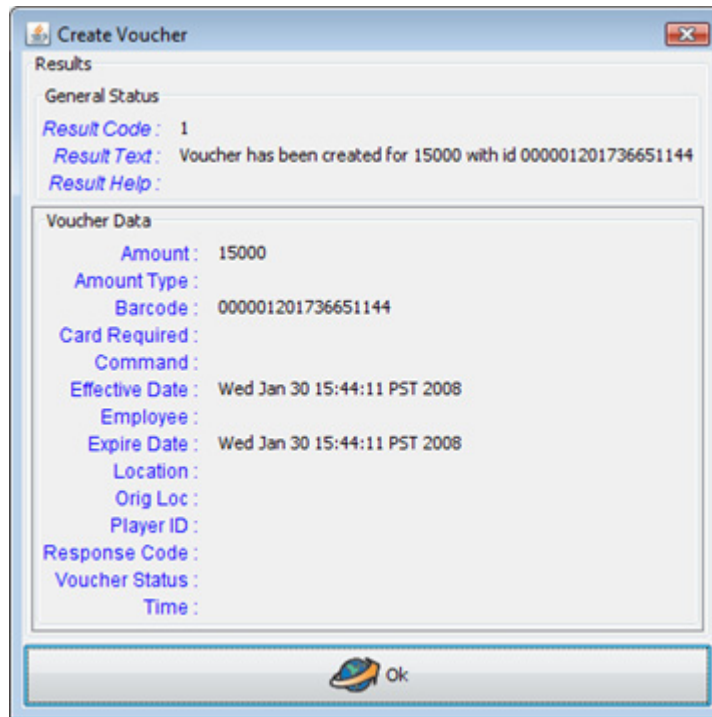
### Create Voucher

1. Click **Create Voucher**.



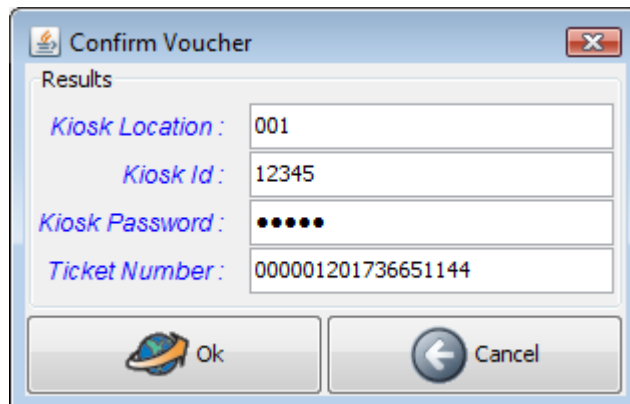
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter a voucher number (**Machine Ticket Number**) for the voucher you are creating.
6. Enter the dollar amount (**Amount**) of the voucher you want to create.

7. Click **OK** to send a **createVoucher** call.



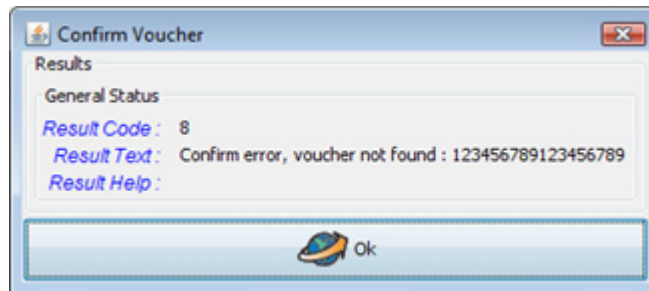
### Confirm Voucher

1. Click **Confirm Voucher**.



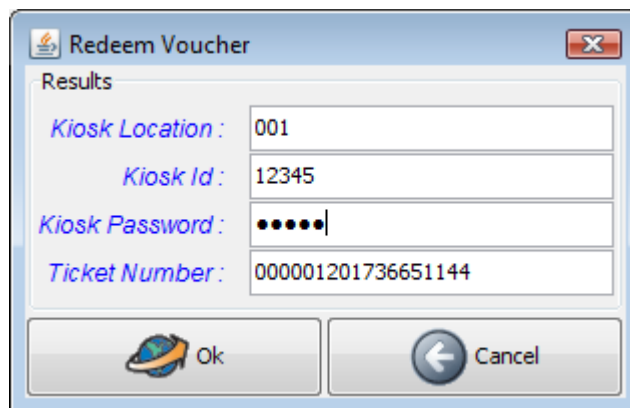
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the **Ticket Number**.

6. Click **OK** to send a **confirmVoucher** call.



### Redeem Voucher

1. Click **Redeem Voucher**.



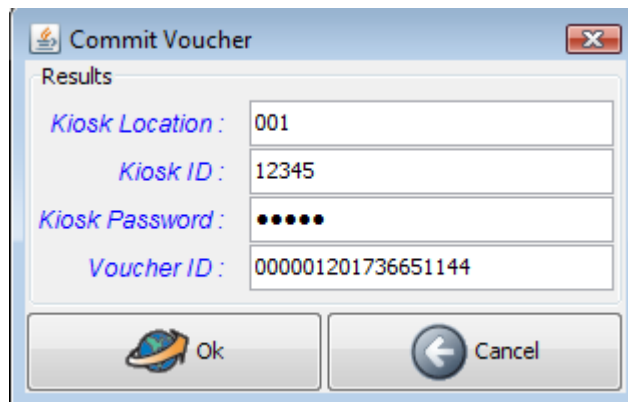
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the **Ticket Number**.

6. Click **OK** to send a **redeemVoucher** call.



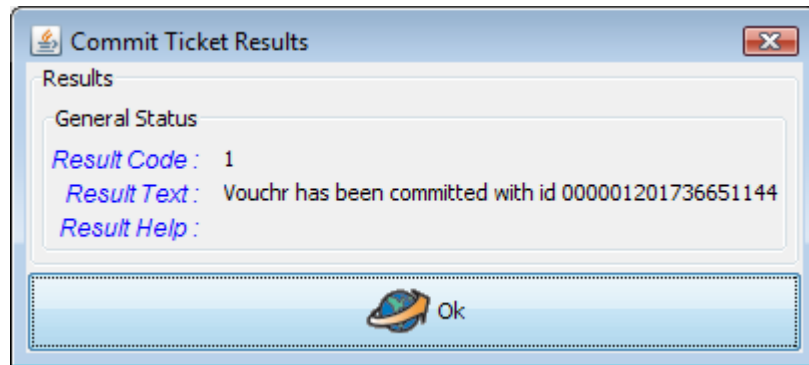
### Commit Voucher

1. Click **Commit Voucher**.



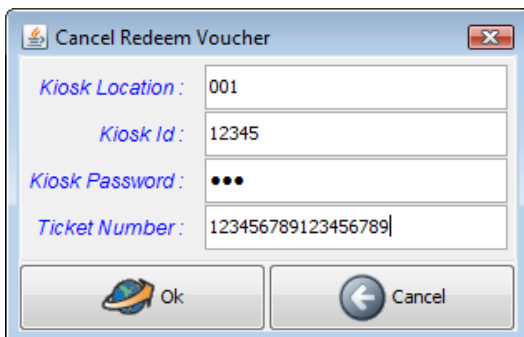
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the **Voucher ID**.

6. Click **OK** to send a **commitVoucher** call.



### Cancel Redeem Voucher

1. Click **Cancel Redeem Voucher** to cancel a Redeem Voucher operation.



Cancel Redeem Voucher

Kiosk Location : 001

Kiosk Id : 12345

Kiosk Password : ...

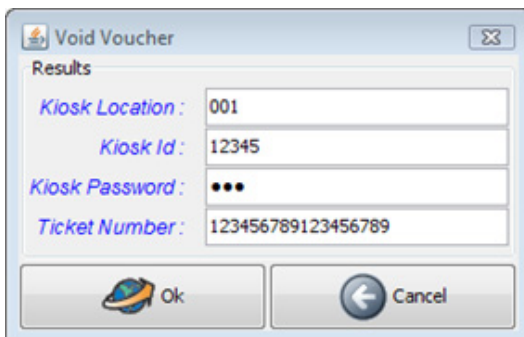
Ticket Number : 123456789123456789

Ok Cancel

2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the **Ticket Number**.
6. Click **OK** to send a **cancelRedeem** call.

### Void Voucher

1. Click **Void Voucher**.



Void Voucher

Results

Kiosk Location : 001

Kiosk Id : 12345

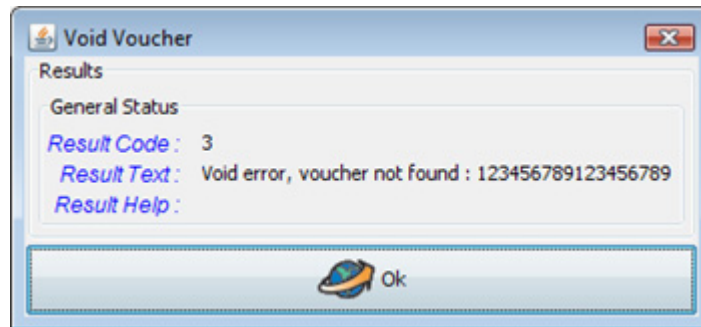
Kiosk Password : ...

Ticket Number : 123456789123456789

Ok Cancel

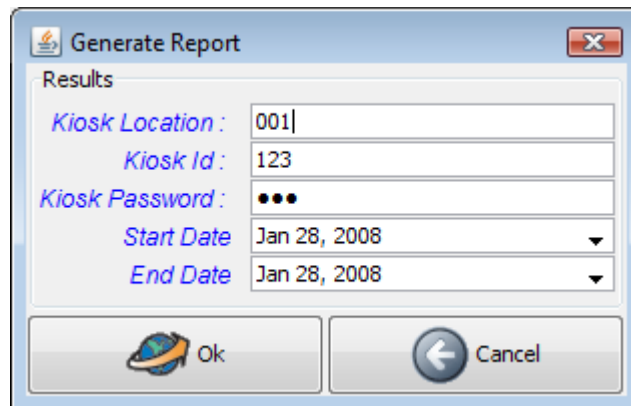
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the **Ticket Number**.
6. Click **OK** to send a **voidVoucher** call.

7. Click **OK** to send a **voidVoucher** call.



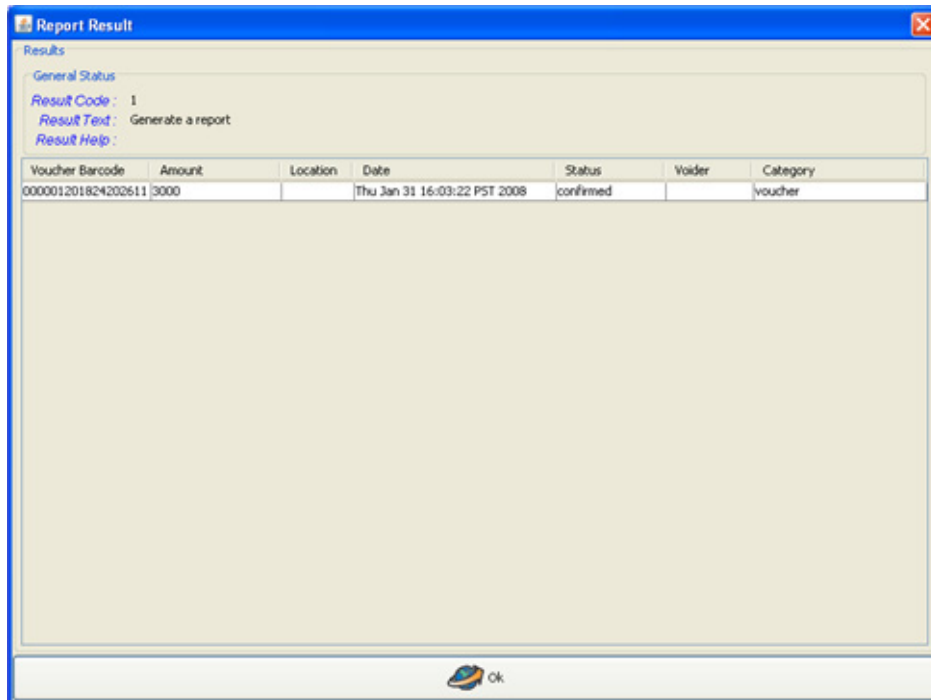
### Generate Report

1. Click **Generate Report**



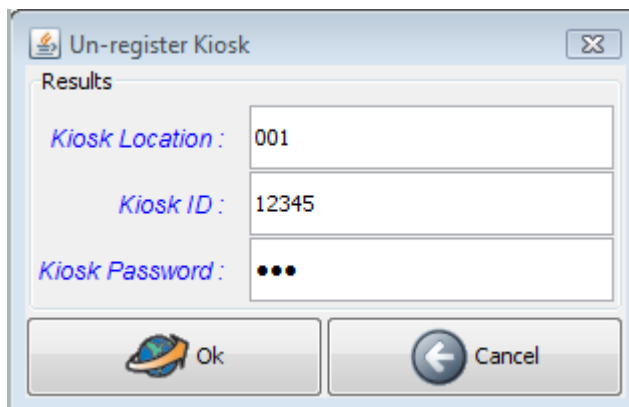
2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.
5. Enter the report **Start Date**.
6. Enter the report **End Date**.

7. Click **OK** to send a **generateReport** call. The following screen displays the results of the call:



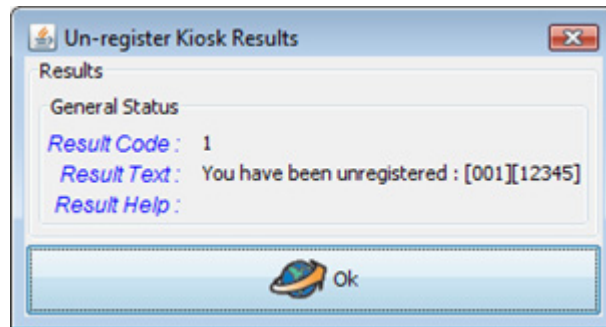
### Unregister Kiosk

1. Click **Unregister Kiosk**.



2. Enter the **Kiosk Location**.
3. Enter the **Kiosk ID**.
4. Enter the **Kiosk Password**.

5. Click **OK** to send an **unregisterKiosk** call.



## About the Mock Host Simulator

The mock host simulator allows you to send messages to the RPC (using either the kiosk simulator or your kiosk implementation), and to view how a voucher system would handle those messages. When you start the RPC, which points to the mock host simulator, a logging window appears. This window shows you all messages sent between the RPC and the mock host simulator.

### Using the Logging Window

When the RPC is started, the RPC logging window displays messages as they are sent between the RPC and voucher system. Use this information to look for implementation errors, and to view traffic between the RPC and voucher system(s).

### Message Format

```
2008-01-17 15:06:30,564 [pool-1-thread-5] INFO - {general} Client asked if TITO is active.
```

<b>2008-01-17 15:06:17, 564</b>	<b>[pool-1-thread-3]</b>	<b>INFO</b>	<b>{general}</b>	<b>Client asked if TMUX is active.</b>
Date and time the log message was printed. Note that the time format is:  HH: MM: SS, MS	Application thread that sent the message.	The type of information in the message: INFO, WARN, DEBUG, ERROR, FATAL.	Message category (currently only "general")	Log message indicating what is happening or happened.



# Chapter 4

## Voucher Service API

---

### Voucher Service Calls

#### connectToVoucherService

##### Purpose

To initiate a connection between RPC and the voucher system.

##### Parameters

Field	Type	Description
configString	String	The configuration string to be used to initialize the connection to the voucher server. See <i>Example Configuration Strings</i> .
enableTransactionLogging	Boolean	Controls whether the RPC enters transactions into its local database - true or false.

##### Example Configuration Strings

Example MOCK Configuration String - Below is an example of the string to send to the MOCK plug-in. This example has no effect other than to provide a sample implementation.

```
empty:empty
```

##### Example STC Configuration String

```
10.1.212.25:8443:5000:5000
stc-sever-ip-address:stc-server-port-number:http-timout:socket-timout
```

##### Example CTN Configuration String

```
192.168.254.22|40666|QUIKPLAY|49160
ctn-server-ip-address|ctn-server-port-number|ctn-type|ctn-id
```

##### Example S2S Configuration String

```
http://localhost:28752/RQS/api-services/S2SAP Ihttp://localhost:38101/
RST/api-services/S2SAPI from-system-url to-system-url
```

**Return**

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## isRpcActive

### Purpose

To verify that the RPC service is running and active. It can be called anytime.

### Parameters

None

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## isVoucherSystemActive

### Purpose

To verify whether the configured voucher service is active. It can be called anytime.

### Parameters

None

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## registerKiosk

### Purpose

To inform RPC that a kiosk is active. This call is used when a kiosk starts up.

### Parameters

Field	Type	Description
propertyId	String	Name of the property or casino location. The RPC accepts strings of any format and length.
kioskId	String	Global, unique identification number for the kiosk. The RPC accepts strings of any format and length.
password	String	Password for the specified kiosk. The RPC accepts strings of any format and length.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## unregisterKiosk

### Purpose

To inform RPC that a kiosk is being taken offline. This call informs RPC that a particular kiosk has become inactive. Once this call is sent, **registerKiosk**, must be sent before RPC will accept calls from the kiosk again.

### Parameters

Field	Type	Description
propertyId	String	Name of the property or casino location. The RPC accepts strings of any format and length.
kioskId	String	Global, unique identification number for the kiosk. The RPC accepts strings of any format and length.
password	String	Password for the specified kiosk. The RPC accepts strings of any format and length.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## createVoucher

### Purpose

To create a new voucher. This call should be followed by either **confirmVoucher** or **voidVoucher**.

### Parameters

Field	Type	Description
propertyId	String	Name of the property or casino location.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
amount	Integer	Amount of the voucher, in cents. Examples: \$100.00 = 10000 \$24.50 = 2450
machineNumber	Integer	User-defined kiosk identification number. This number allows the client side to set its own tracking numbers. Most systems allow values from 1 to 9999, depending on the system.

### Return

This method returns a [voucherServiceResult](#) object. The object contains a voucher entry. This entry contain all of the fields that can appear on the physical voucher.

```
resultCode  
    0 - Failed  
    1 - Success
```

## confirmVoucher

### Purpose

To confirm that a voucher was successfully printed. This call should follow a **createVoucher** call.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
voucherId	String	Voucher identification number to confirm.
amount	Integer	Amount of the voucher, in cents. Examples: \$100.00 = 10000 \$24.50 = 2450
machineNumber	Integer	User-defined kiosk identification number. This number allows the client side to set its own tracking numbers. Most systems allow values from 1 to 9999, depending on the system.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## voidVoucher

### Purpose

To mark a voucher as invalid. This call should be sent after **createVoucher** if a voucher was *not* printed successfully.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
voucherId	String	Voucher identification number to void.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## redeemVoucher

### Purpose

To redeem a voucher. This call must be followed by a **commitVoucher** or a **cancelRedeem** call.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
voucherId	String	Voucher identification number to redeem.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## cancelRedeem

### Purpose

To cancel a redemption operation. This call must be sent after a **redeemVoucher** call.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
redemptionData	<a href="#">voucherTicketRedemptionData</a>	Voucher identification number to redeem.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## commitVoucher

### Purpose

To indicate that a kiosk has physically redeemed a voucher. This call should be sent after a **redeemVoucher** call.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
redemptionData	<a href="#">voucherTicketRedemptionData</a>	Voucher identification number redeemed.

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## generateReport

### Purpose

To generate a report of kiosk activity. It can be called anytime after a **registerKiosk** call.

### Parameters

Field	Type	Description
propertyId	String	Location of the specified kiosk.
kioskId	String	Global, unique identification number for the kiosk.
password	String	Password for the specified kiosk.
startDate	Date	Beginning date and time of requested activity, in the format: Day Mon DD HH:MM:SS TMZ YYYY. Example: Wed Jan 23 13:30:36 PST 2008
endDate	Date	Ending date and time of requested activity, in the format: Day Mon DD HH:MM:SS TMZ YYYY. Example: Wed Jan 23 13:30:36 PST 2008

### Return

This method returns a [voucherServiceResult](#) object. Within the object is an array of [voucherReportEntry](#).

```
resultCode  
    0 - Failed  
    1 - Success
```

## verifyChecksum

### Purpose

To verify that the RPC server is an unaltered server.

### Parameters

Field	Type	Description
Type	String	MD5

### Return

This method returns a [voucherServiceResult](#) object that contains the current MD5 verification sequence. You can look at the checksum member of the **voucherServiceResult** object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## disconnectFromVoucherService

### Purpose

To sever a connection between RPC and the voucher system.

### Parameters

Field	Type	Description
configString	String	<i>For future use.</i>

### Return

This method returns a [voucherServiceResult](#) object.

```
resultCode  
    0 - Failed  
    1 - Success
```

## Call Responses

### **voucherServiceResult**

Every API call returns a **voucherServiceResult** object. This object contains the result information for the various API calls.

The object contains the following result fields:

Field	Type	Description
checksum	String	Result of checksum operation.
redemptionData	<a href="#">voucherTicketRedemptionData</a>	Result of a redemption request. This field can also be used as a parameter in the <b>commitVoucher</b> request.
report	<a href="#">voucherReportEntry</a>	Result of generate report operation.
resultCode	Integer	Error code. For more information, see <i>SOAP Errors on page B-1</i> .
resultHelp	String	Additional error information intended to assist issue resolution.
resultText	String	Description of specified error code.
voucher	<a href="#">voucherTicketData</a>	If <b>createVoucher</b> , this field contains voucher data.

## voucherTicketData

When the **createVoucher** API call is issued, a **voucherTicketData** object is returned as part of the **voucherServiceResult** object. The fields returned in this call depend on the protocol used by the voucher system.

Field	Type	Description
amount	String	Amount of the voucher, in pennies.
amountType	String	Indicates whether the voucher is promotional or non-promotional, cashable or non-cashable.
barcode	String	18-digit voucher barcode.
cardRequired	String	“True” indicates a player card is required to redeem the voucher.
command	String	Command that was sent to the server.
effectiveDate	String	Date and time a voucher becomes eligible for redemption. The format is: Day Mon DD HH:MM:SS TMZ YYYY.  Example: Wed Jan 23 13:30:36 PST 2008
employee	String	Employee identifier associated with the specified voucher status.
expireDate	String	Date and time a voucher is no longer eligible for redemption. The format is: Day Mon DD HH:MM:SS TMZ YYYY.  Example: Wed Jan 23 13:30:36 PST 2008
location	String	Voucher status.
machineNumber	String	User-defined kiosk identification number. This number allows the client side to set its own tracking numbers. Most systems allow values from 1 to 9999, depending on the system.
orig_loc	String	Location where the voucher was created.
playerid	String	Player identifier associated with the specified voucher. While the typical format is a 10-digit number, the RPC accepts any string, of any length.
propertyAddr1	String	First line of property address.
propertyAddr2	String	Second line of property address.

Field	Type	Description
propertyName	String	Property name.
response_code	String	Voucher system response.
ticketStatus	String	Status of the voucher in the database. <i>See the voucher system vendor for a status list.</i>
time	String	Time voucher was created in the voucher system.
titleCash	String	Title printed on vouchers for cashable credits.
titleLargeWin	String	Title printed on vouchers for large-win credits.
titleNonCash	String	Title printed on vouchers for non-cashable credits.
titleNonPromo	String	Title printed on vouchers for non-promotional credits.
titlePromo	String	Title printed on vouchers for promotional credits.

## voucherReportEntry

When the **generateReport** API call is issued, an array **voucherReportEntry** object is returned as part of the **voucherServiceResult** object. The fields returned in this object depend on the protocol.

Field	Type	Description
amount	String	Amount of voucher, in pennies.
barcode	String	18-digit voucher barcode.
category	String	<i>For future use.</i>
dateTime	String	Date and time that the voucher was issued. The format is: Day Mon DD HH:MM:SS TMZ YYYY. Example: Wed Jan 23 13:30:36 PST 2008
location	String	Location identification number of the kiosk where the voucher was requested.
status	String	Status of the voucher: confirmed, committed, voided, redeemed and created.
voider	String	Identification

**voucherTicketRedemptionData**

Field	Type	Description
amount	String	Amount of voucher.
amountType	String	Currency code. Example: United States dollar = USD See <i>ISO 4217</i> for a list of currency codes by country.
barcode	String	18-digit voucher barcode.
cardRequired	String	“True” indicates a player card is required to redeem the voucher.
cardRestricted	String	Indicates player card restrictions: no card, any card or this card.
creditType	String	Indicates fund type: cashable, non-cashable, promotional, promoPool or nonCashPool.
effectiveDate	String	Date and time a voucher is eligible for redemption. The format is: Day Mon DD HH:MM:SS TMZ YYYY. Example: Wed Jan 23 13:30:36 PST 2008
employee	String	Employee identifier associated with the specified voucher status.
expireDate	String	Date and time a voucher is no longer eligible for redemption. The format is: Day Mon DD HH:MM:SS TMZ YYYY. Example: Wed Jan 23 13:30:36 PST 2008
largeWin	String	Used by an EGM to send handpay information. The handpay information includes the handpay amount of the handpay and whether it can be paid remotely.
location	String	Location where the voucher was redeemed.
machineTicket	String	
playerId	String	Player identifier associated with the specified voucher. While the typical format is a 10-digit number, the RPC accepts any string, of any length.
poolId	String	Pool for promotional or non-cashable credits.

Field	Type	Description
responseString	String	Indicates whether the voucher status is "Successful" or displays the reason for failure.
ticketStatus	String	Status of the voucher in the database. See the voucher system vendor for a status list.
transactionId	String	Voucher transaction identifier.
transferAction	String	Type of action taken (for example, create or redeem).
transferAmount	String	Actual amount transferred.
transferCode	String	01 - Valid restricted promotional ticket 02 - Valid nonrestricted promotional ticket 80 - Unable to validate 81 - Not a valid validation number 82 - Validation number not in system 83 - Ticket marked pending in the system 84 - Ticket already redeemed 85 - Ticket expired 86 - Validation Information not available 87 - Ticket amount doesn't match system amount 88 - Ticket amount exceeds auto redemption limit FF - Request for ticket status



# Appendix A

## Example Configuration Files

---

For information on modifying configuration files, see *Modify Configuration File(s)* on page 3-2.

### RPC\_CONFIG MOCK.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<configuration>
  <rpc>
    <id>RPC-MOCK</id>
  </rpc>
  <derby>
    <drda>
      <portNumber>42301</portNumber>
    </drda>
    <system>
      <home>../database/rpc/mock</home>
    </system>
  </derby>
  <tito>
    <server>
      <type>MOCK-v1.0</type>
      <setup>empty:empty</setup>
    </server>
  </tito>
  <voucher>
    <service>
      <port>http://localhost:8080/Multiplexer/VoucherService</port>
    </service>
  </voucher>
  <management>
    <service>
      <port>http://localhost:8080/Multiplexer/ManagementService</port>
    </service>
  </management>
</configuration>
```

## RPC\_CONFIG\_CTN.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<configuration>
  <rpc>
    <id>RPC-CTN</id>
  </rpc>
  <derby>
    <drda>
      <portNumber>42304</portNumber>
    </drda>
    <system>
      <home>../database/rpc/ctn</home>
    </system>
  </derby>
  <tito>
    <server>
      <type>CTN</type>
      <setup>192.168.254.22|40666|QUIKPLAY|49160</setup>
    </server>
  </tito>
  <voucher>
    <service>
      <port>http://localhost:8080/Multiplexer/VoucherService</port>
    </service>
  </voucher>
  <management>
    <service>
      <port>http://localhost:8080/Multiplexer/ManagementService</port>
    </service>
  </management>
</configuration>
```

## RPC\_CONFIG\_S2S1-2-6.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<configuration>
  <rpc>
    <id>RPC-S2S1.2.6</id>
  </rpc>
  <derby>
    <drda>
      <portNumber>42302</portNumber>
    </drda>
    <system>
      <home>../database/rpc/s2s</home>
    </system>
  </derby>
  <tito>
    <server>
      <type>S2S-1.2.6</type>
      <setup>http://localhost:28752/RQS/api-services/S2SAPI
http://localhost:38101/RST/api-services/S2SAPI</setup>
    </server>
  </tito>
  <voucher>
    <service>
      <port>http://localhost:8080/Multiplexer/VoucherService</port>
    </service>
  </voucher>
  <management>
    <service>
      <port>http://localhost:8080/Multiplexer/ManagementService</port>
    </service>
  </management>
</configuration>
```

## RPC\_CONFIG\_STC9X.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<configuration>
  <rpc>
    <id>RPC-STC</id>
  </rpc>
  <derby>
    <drda>
      <portNumber>42303</portNumber>
    </drda>
    <system>
      <home>../database/rpc/stc</home>
    </system>
  </derby>
  <tito>
    <server>
      <type>STC-9.x</type>
      <setup>10.1.217.111:8443</setup>
    </server>
  </tito>
  <voucher>
    <service>
      <port>http://localhost:8081/Multiplexer/VoucherService</port>
    </service>
  </voucher>
  <management>
    <service>
      <port>http://localhost:8081/Multiplexer/ManagementService</port>
    </service>
  </management>
</configuration>
```



## Appendix B

### SOAP Errors

---

Error Code	Description
1	Command executed successfully. public static final int VOUCHER_SERVICE_OK
6	Unable to create voucher. public static final int VOUCHER_SERVICE_VOUCHER_CREATE_ERROR
8	Unable to confirm voucher. public static final int VOUCHER_SERVICE_VOUCHER_CONFIRM_ERROR
10	There was a void error. public static final int VOUCHER_SERVICE_VOUCHER_VOID_ERROR
12	There was a problem generating the voucher report. public static final int VOUCHER_SERVICE_GENERATE_REPORT_ERROR
14	There was a general HTTP communication error. public static final int VOUCHER_SERVICE_HTTP_ERROR
16	There was a general input/output (I/O) error. public static final int VOUCHER_SERVICE_IO_ERROR
18	The voucher service reported an error. See error messaging text. public static final int VOUCHER_SERVICE_TITO_ERROR
20	The voucher system is in an invalid state. public static final int VOUCHER_SERVICE_VOUCHER_INVALID_STATE

Error Code	Description
32	Property ID is required. public static final int VOUCHER_SERVICE_PROPERTY_ID_REQUIRED
34	Kiosk ID is required. public static final int VOUCHER_SERVICE_KIOSK_ID_REQUIRED
36	Password is required. public static final int VOUCHER_SERVICE_PASSWORD_REQUIRED
38	Voucher ID is required. public static final int VOUCHER_SERVICE_VOUCHERID_REQUIRED
40	A starting date is required for the completion of the command. public static final int VOUCHER_SERVICE_START_DATE_REQUIRED
42	An ending date is required for the completion of the command. public static final int VOUCHER_SERVICE_END_DATE_REQUIRED
50	The specified kiosk was not found. public static final int VOUCHER_SERVICE_KIOSK_NOT_FOUND
100	The method called is not implemented. public static final int VOUCHER_SERVICE_NOT_IMPLEMENTED
102	The voucher service is not ready. public static final int VOUCHER_SERVICE_NOT_READY
200	Voucher amount limit exceeded. public static final int VOUCHER_SERVICE_VOUCHER_AMOUNT_LIMIT_EXCEEDED
202	General error. See error message text. public static final int VOUCHER_SERVICE_VOUCHER_GENERAL_ERROR
204	The specified voucher was not found. public static final int VOUCHER_SERVICE_VOUCHER_NOT_FOUND
302	There was an error when processing the checksum for the CORE RPC. public static final int VOUCHER_SERVICE_CORE_CHECKSUM_ERROR
304	There was a problem generating the checksum for the plug-in. public static final int VOUCHER_SERVICE_PLUGIN_CHECKSUM_ERROR

---

Error Code	Description
500	There was a redeem error. <code>public static final int VOUCHER_SERVICE_VOUCHER_REDEEM_ERROR</code>
600	The specified operation is not supported by the voucher system. <code>public static final int VOUCHER_SERVICE_OPERATION_NOT_SUPPORTED</code>





# Index

---

## A

API  
    flow diagrams 2-1  
    stateless 1-1  
Application Protocol Interface, *See* API 1-1  
Aristocrat GSA protocol 1-5  
authentication 3-4

## B

Bally Technologies STC protocol 1-5

## C

cacerts 3-4  
cancel redeem voucher 3-16  
cancelRedeem 2-6  
commit voucher 3-14  
commitVoucher 2-6  
config files 3-2  
configuration files  
    attributes 3-3  
    modify 3-2  
confirm voucher 3-12  
confirmVoucher 1-4, 2-4  
connectToVoucherSystem 1-4, 2-1  
create voucher 3-11  
createTicket 3-11  
createVoucher 1-4, 2-3

## D

disconnectFromVoucherSystem 2-8

## E

encryption 3-4

## G

generateReport 1-4, 2-7, 3-17

## H

host simulator  
    procedure 3-4  
    purpose 3-20

## I

IGT CTN protocol 1-5  
initialize connection 3-8  
isRPCActive 1-4  
isRpcActive 1-5, 2-2  
isTitoSystemActive 1-4, 2-2, 3-9  
isTmuxActive 3-9

- 
- J**
- Java 1-5
- K**
- kiosk client simulator
    - purpose 3-6
    - using 3-7
  - kioskId 1-5
- L**
- logging window 3-20
- M**
- mock host simulator 3-20
- P**
- password 1-5
  - propertyId 1-5
- R**
- redeem voucher 3-13
  - redeemVoucher 1-4, 2-5, 2-6
  - register kiosk 3-10
  - registerKiosk 1-4, 2-1, 3-10
  - RPC
    - developing an interface 3-1
    - development roadmap 3-1
    - getting started 3-1
    - installation 3-2
  - RPC checksum 3-10
  - RPC instance 1-2
    - single 1-2
  - RPC instances
    - multiple 1-2
  - RPC server 1-2
    - dedicated 1-2
    - requirements 1-5
    - shared 1-2
    - shared server flow diagram 1-2
    - single server flow diagram 1-2
  - rpc\_config\_ctn.xml 3-2
  - rpc\_config\_mock.xml 3-2
  - rpc\_config\_s2s1-2-6.xml 3-2
  - rpc\_kiosk\_client.bat 3-6
- S**
- Simple Object Access Protocol, *See SOAP* 1-1
  - simulator
    - flow diagram 1-3
    - host 1-3, 3-20
    - kiosk client 1-3
  - SOAP 1-1
- U**
- unregister kiosk 3-18
  - unregisterKiosk 1-4, 2-7
- V**
- verify RPC 3-9
  - verify voucher system 3-9
  - verifyChecksum 1-4, 1-5, 3-10
  - void voucher 3-16
  - voidVoucher 1-4, 2-4
  - voucher service call
    - verifyChecksum 1-4
  - voucher service call responses
    - voucherReportEntry 2-7
    - voucherServiceResult 2-3, 2-5
    - voucherTicketData 2-3

voucher service calls

- cancelRedeem 2-6
- commitVoucher 2-6
- confirmVoucher 1-4, 2-4
- connectToVoucherSystem 1-4, 2-1
- createVoucher 1-4, 2-3
- disconnectFromVoucherSystem 2-8
- generateReport 1-4, 2-7
- isRPCActive 1-4
- isRpcActive 2-2
- isTitoSystemActive 1-4
- isVoucherSystemActive 2-2
- redeemVoucher 1-4, 2-5, 2-6
- registerKiosk 1-4, 2-1
- unregisterKiosk 1-4, 2-7
- voidVoucher 1-4, 2-4

voucher systems

- supported 1-5
- voucherReportEntry 2-7
- voucherServiceResult 2-3, 2-5
- voucherTicketData 2-3

**W**

- Windows XP Professional 1-5

