Chapter 1

Look Inside

Introduction

1.1 Overview

This document describes the Simple System Interface. The Simple System Interface is designed to allow systems to exchange information with other systems in a very simple manner.

With the Simple System Interface, a Client System uses the HTTP protocol and HTTP verbs to access resources on a Host System. The data exchanged between systems is encoded using JSON.

The resources on the Host System could allow the Client System to retrieve (HTTP GET) information from the Host System. For example, a G2S host could act as a Client System and request information about players from a Host System. Or, the resources on the Host System could allow the Client System to report (HTTP POST) information to the Host System. For example, a G2S host could act as a Client System and report events, which are originated by EGMs, to a data warehouse acting as a Host System. Other paradigms and uses of the HTTP verbs are possible.

The resources available through the Simple System Interface are described in subsequent chapters of this document.

.....*(continued)*...

Chapter 4

Look Inside

Voucher Resources

Extension in v1.1

NEW CHAPTER

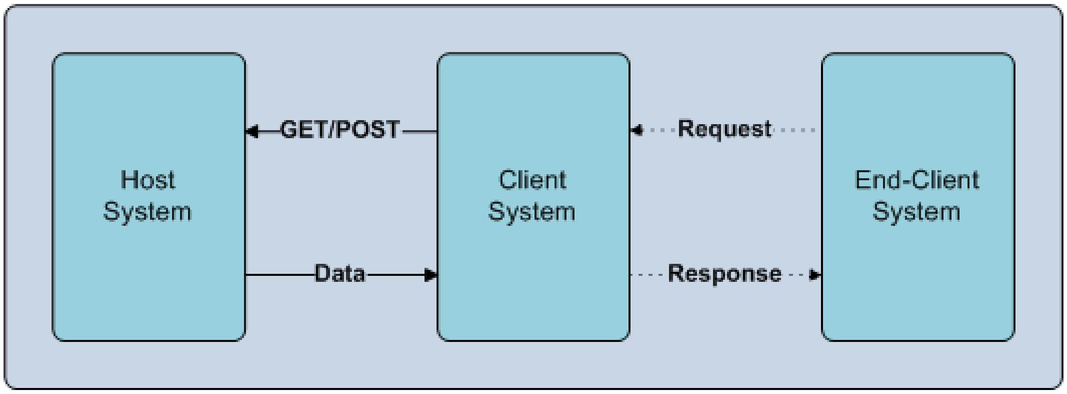
4.1 Introduction

The resources within this chapter are used to manage the issuance and redemption of payment vouchers by end-clients, such as redemption kiosks or cashier terminals. Payment vouchers are sometimes referred to as "tickets" or "coupons".

For example, the resources in this chapter can be used to record the issuance of a voucher by an end-client. Subsequently, other resources in this chapter can be used to authorize the redemption of the voucher by the same end-client or a different end-client.

Resources within this chapter can also be used to manage the configuration options used by end-clients when performing voucher processing operations.

The resources are designed so that a client system could be acting on behalf of a series of end-clients, passing requests from the end-clients through to the host. In the simplest case, the client system can be the end-client itself, making requests on its own behalf. In this chapter, it is assumed that the client is acting on behalf of a series of end-clients. The simpler case, where the end-client is acting on its own behalf, is also possible.



This functionality maps directly to similar functionality within the G2S and S2S protocols allowing a central system to easily manage voucher processing operations across a series of clients using the G2S and/or S2S protocols, as well as SSI.

4.1.1 Sequence Diagrams

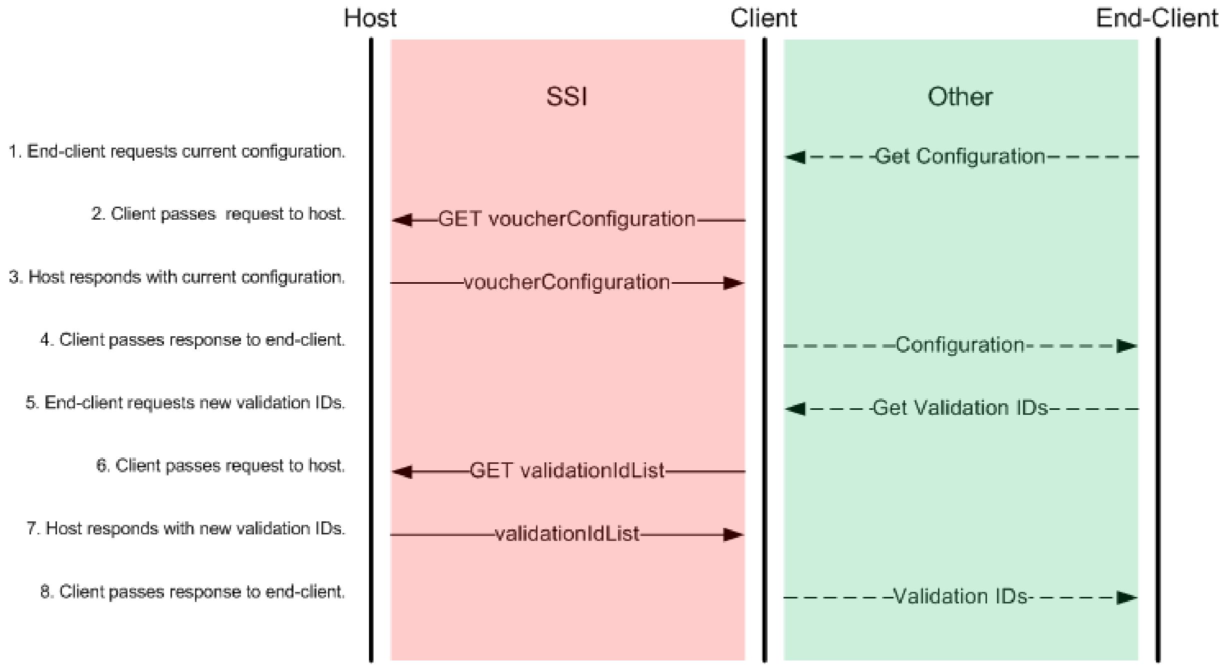
The following sequence diagrams demonstrate how the resources within this chapter are intended to be used to manage the issuance and redemption of vouchers as well as the configuration of end-clients. Other scenarios are possible.

4.1.1.1 Initial Configuration

The following sequence diagram demonstrates the expected behavior when an end-client first starts communicating with the host or resumes communicating after an outage.

1. The end-client requests the current configuration options from the client.
2. The client passes the request through to the host.
3. The host responds to the client with the current configuration options.
4. The client passes the current configuration options through to the end-client.
5. The end-client requests a new set of validation identifiers from the client.

6. The client passes the request through to the host.
7. The host responds with a new set of validation identifiers.
8. The client passes the new set of validation identifiers through to the end-client.


Host
Client
End-Client
SSI
Other
1. End-client requests current configuration.
2. Client passes request to host.
3. Host responds with current configuration.
4. Client passes response to end-client.
5. End-client requests new validation IDs.
6. Client passes request to host.
7. Host responds with new validation IDs.
8. Client passes response to end-client.
Get Configuration
GET voucherConfiguration
voucherConfiguration
Configuration
Get Validation IDs
GET validationIdList
validationIdList
Validation IDs

4.1.1.2 Configuration Changes

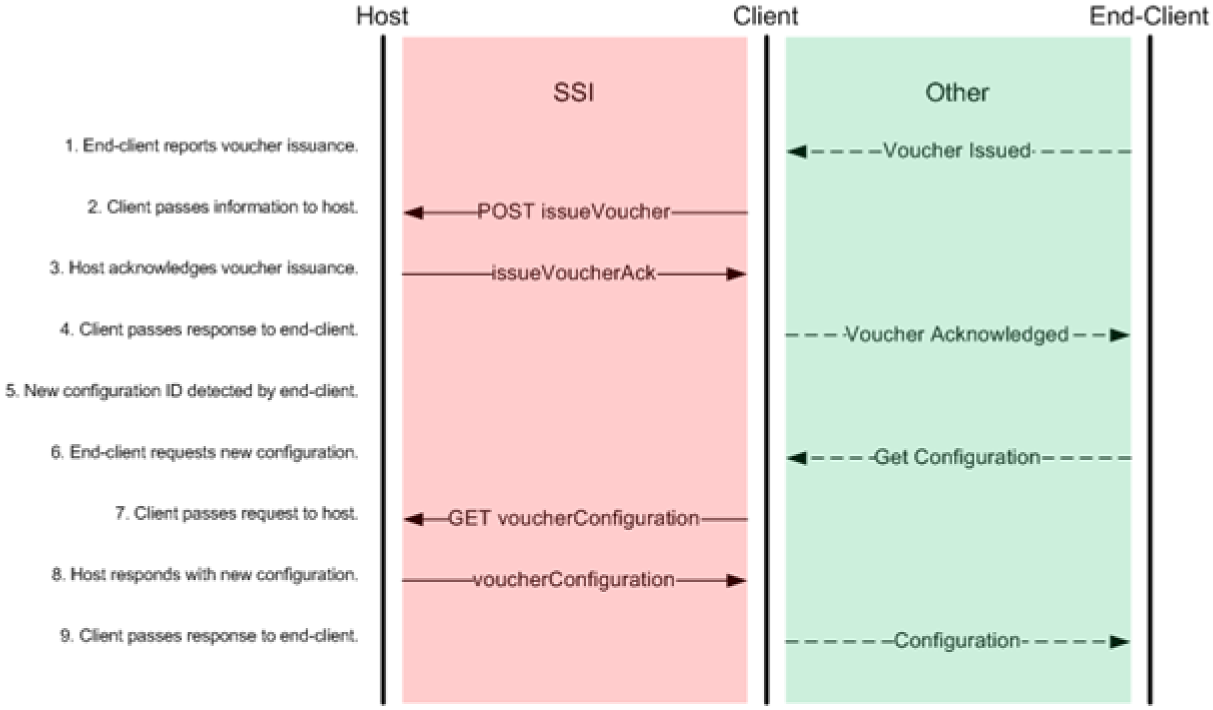
The following sequence diagram demonstrates the expected behavior when an end-client detects that the configuration options have changed and that a new set of configuration options is needed.

Each response from the host contains a configuration identifier. The configuration identifier identifies the current set of configuration options that should be used by the end-client. The end-client is expected to compare the configuration identifier contained in the responses from the host to the configuration identifier currently being used by the end-client. If the configuration identifiers are not the same, the end-client is expected to request a new set of configuration options.

In this example, the end-client detects a change to the configuration identifier in the acknowledgement to a voucher issuance request. The change could have been detected in other responses as well.

1. The end-client notifies the client that a voucher has been issued.
2. The client passes the voucher issuance request through to the host.
3. The host acknowledges the voucher issuance request.
4. The client passes the acknowledgement through to the end-client.
5. The end-client detects that the configuration identifier contained in the acknowledgement is different than the configuration identifier for the current configuration options that it is using.
6. The end-client requests the current configuration options from the client.
7. The client passes the request through to the host.
8. The host responds to the client with the current configuration options.

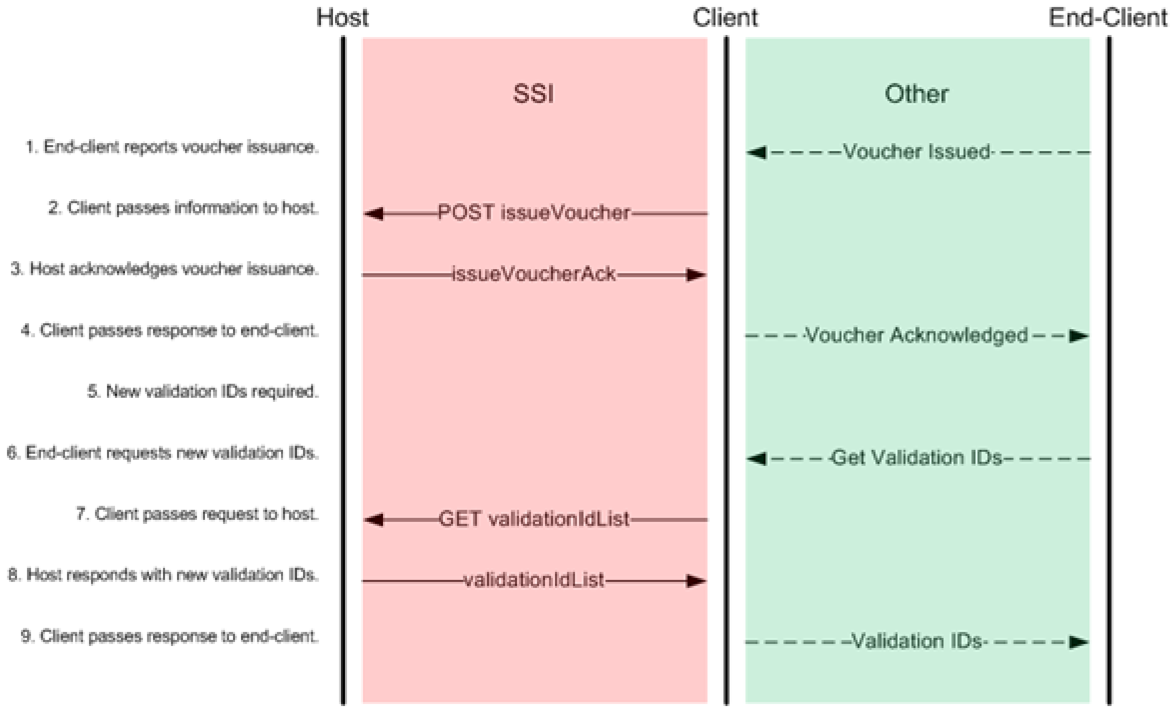
9. The client passes the current configuration options through to the end-client.


Host
Client
End-Client
SSI
Other
1. End-client reports voucher issuance.
2. Client passes information to host.
3. Host acknowledges voucher issuance.
4. Client passes response to end-client.
5. New configuration ID detected by end-client.
6. End-client requests new configuration.
7. Client passes request to host.
8. Host responds with new configuration.
9. Client passes response to end-client.
Voucher Issued
POST issueVoucher
issueVoucherAck
Voucher Acknowledged
Get Configuration
GET voucherConfiguration
voucherConfiguration
Configuration

4.1.1.3 Voucher Issuance

The following sequence diagram demonstrates the expected behavior when an end-client issues a voucher. If the issuance of the voucher causes the supply of validation identifiers to fall below the required limits, the end-client will request a new set of validation identifiers. This operation is shown in the sequence diagram. However, it would only be performed if the supply of validation identifiers fell below the required limits.

1. The end-client notifies the client that a voucher has been issued.
2. The client passes the voucher issuance request through to the host.
3. The host acknowledges the voucher issuance request.
4. The client passes the acknowledgement through to the end-client.
5. The end-client determines that a new set of validation identifiers is needed.
6. The end-client requests a new set of validation identifiers from the client.
7. The client passes the request through to the host.
8. The host responds with a new set of validation identifiers.
9. The client passes the new set of validation identifiers through to the end-client.


Host
Client
End-Client
SSI
Other
1. End-client reports voucher issuance.
2. Client passes information to host.
3. Host acknowledges voucher issuance.
4. Client passes response to end-client.
5. New validation IDs required.
6. End-client requests new validation IDs.
7. Client passes request to host.
8. Host responds with new validation IDs.
9. Client passes response to end-client.
Voucher Issued
POST issueVoucher
issueVoucherAck
Voucher Acknowledged
Get Validation IDs
GET validationIdList
validationIdList
Validation IDs

4.1.1.4 Voucher Redemption

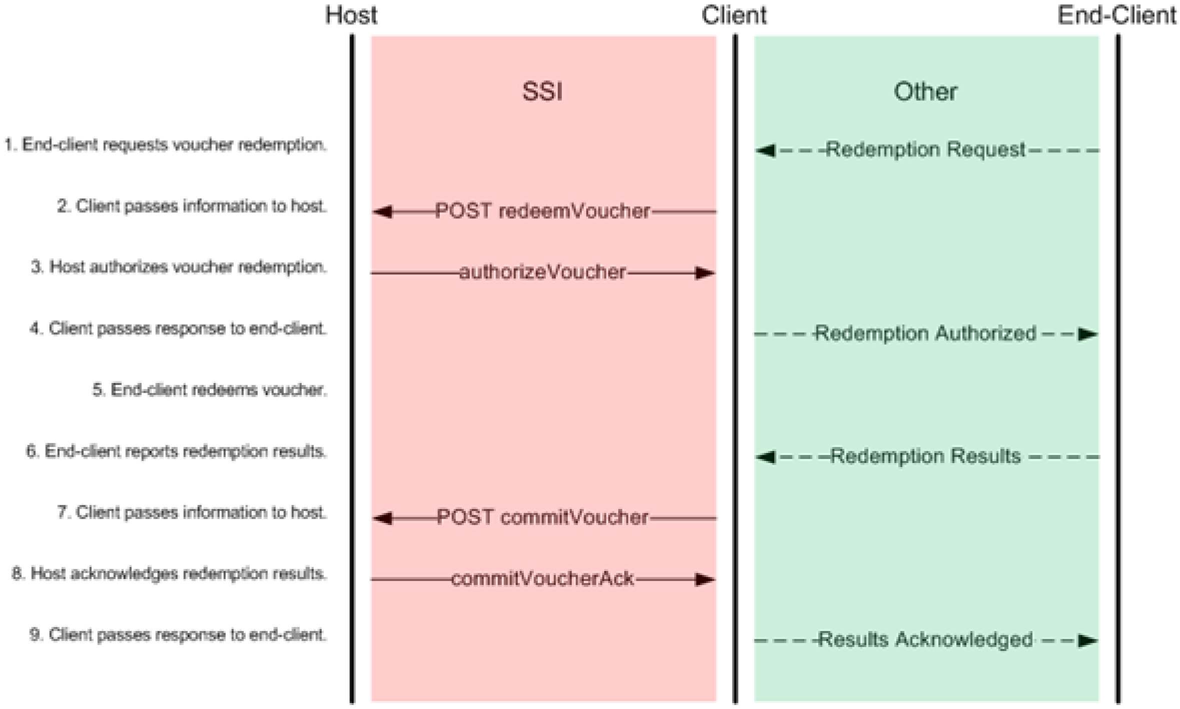
The following sequence diagram demonstrates the expected behavior when an end-client redeems a voucher.

In this example, the host authorizes redemption of the voucher and the end-client redeems the voucher. Alternatively, the host could have denied the redemption request by including a non-zero host exception code in its response. Or, the end-client could have failed to redeem the voucher, in which case a non-zero end-client exception code would have been reported.

Regardless of the outcome, once a redemption request has been made, the end-client must always report the final results of the redemption request even if the request is denied, a host or end-client exception occurs, or the `authorizeVoucher` response is never received.

1. The end-client sends a voucher redemption request to the client.
2. The client passes the voucher redemption request through to the host.
3. The host authorizes the redemption of the voucher.
4. The client passes the authorization through to the end-client.
5. The end-client redeems the voucher.
6. The end-client notifies the client that the voucher has been redeemed.
7. The client passes the redemption notification through to the host.
8. The host acknowledges the redemption notification to the client.
9. The client passes the acknowledgement through to the end-client.

NEW CHAPTER

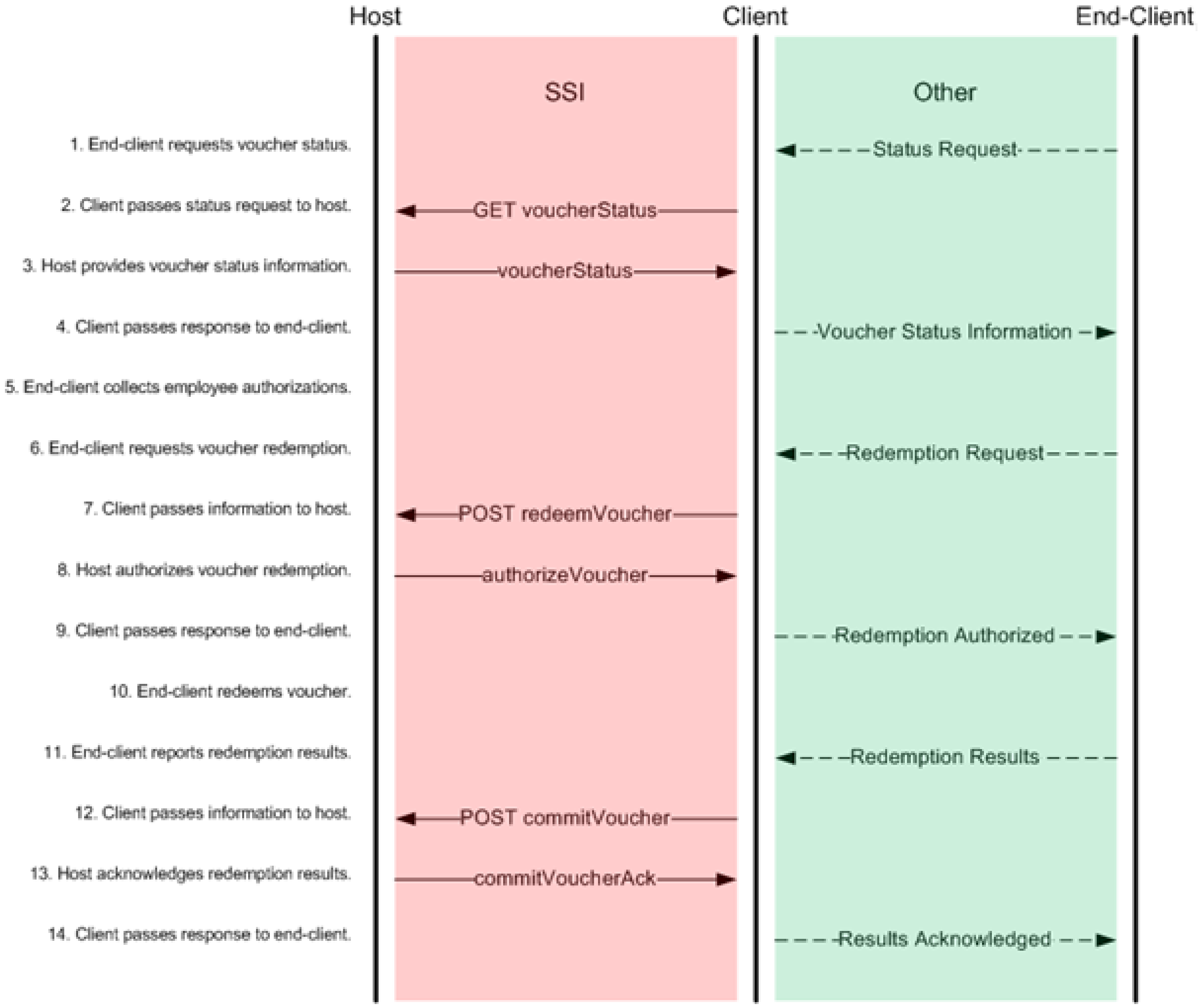

Host
Client
End-Client
SSI
Other
1. End-client requests voucher redemption.
2. Client passes information to host.
3. Host authorizes voucher redemption.
4. Client passes response to end-client.
5. End-client redeems voucher.
6. End-client reports redemption results.
7. Client passes information to host.
8. Host acknowledges redemption results.
9. Client passes response to end-client.
Redemption Request
POST redeemVoucher
authorizeVoucher
Redemption Authorized
Redemption Results
POST commitVoucher
commitVoucherAck
Results Acknowledged

4.1.1.5 Voucher Redemption by Authorized Employees

The following sequence diagram demonstrates the expected behavior when an employee authorization is required before a voucher can be redeemed at an end-client. For example, a supervisor authorization may be required to redeem a large win voucher at a cashier terminal. In this example, the employee authorization information is sufficient and the voucher is redeemed. If the host determined that the employee authorization information was insufficient, the host would have included a non-zero host exception code in its response to the voucher redemption request. Employee authorizations are an optional feature of this specification.

1. The end-client sends a voucher status request to the client.
2. The client passes the voucher status request to the host.
3. The host provides the voucher status information, which includes the employee authorization requirements, to the client.
4. The client passes the voucher status information to the end-client.
5. The end-client collects the required employee authorizations.
6. The end-client sends a voucher redemption request, including the employee authorizations, to the client.
7. The client passes the voucher redemption request through to the host.
8. The host authorizes the redemption of the voucher.
9. The client passes the authorization through to the end-client.
10. The end-client redeems the voucher.
11. The end-client notifies the client that the voucher has been redeemed.

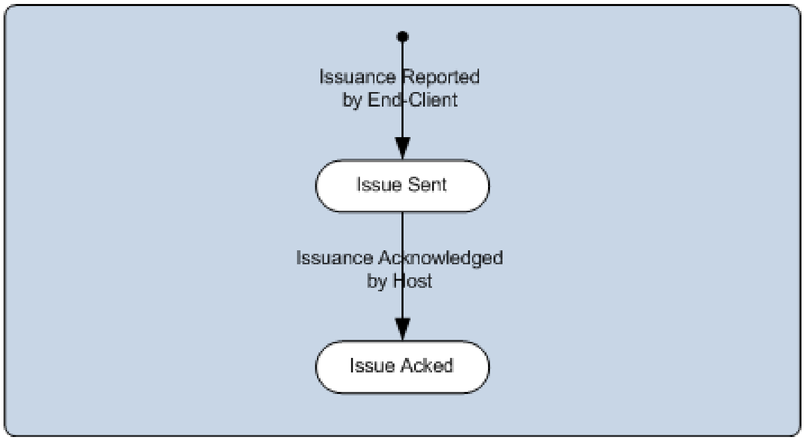
12. The client passes the redemption notification through to the host.
13. The host acknowledges the redemption notification to the client.
14. The client passes the acknowledgement through to the end-client.

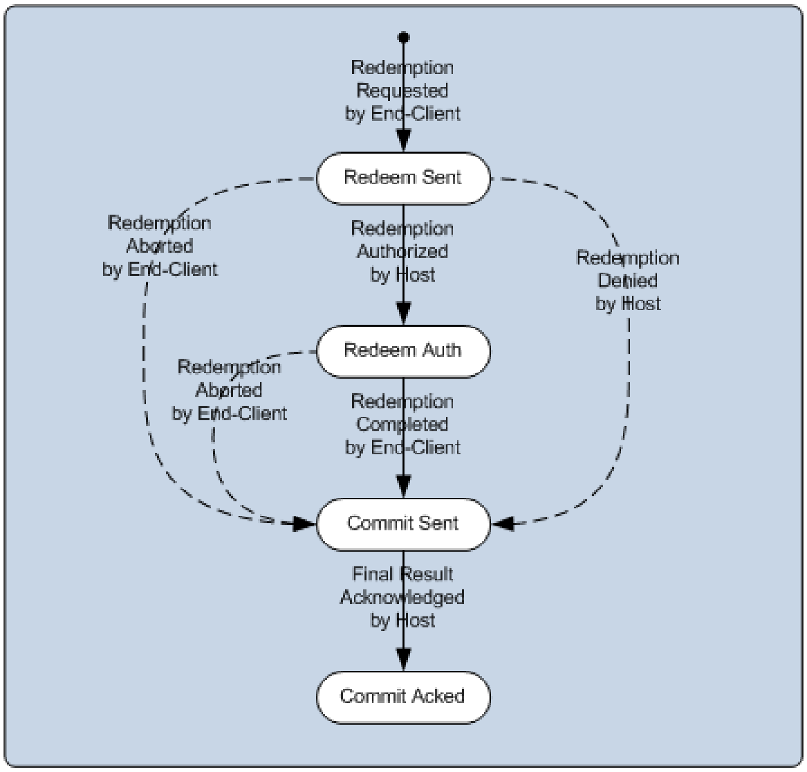

Host
Client
End-Client
SSI
Other
1. End-client requests voucher status.
Status Request
2. Client passes status request to host.
GET voucherStatus
3. Host provides voucher status information.
voucherStatus
4. Client passes response to end-client.
Voucher Status Information
5. End-client collects employee authorizations.
6. End-client requests voucher redemption.
Redemption Request
7. Client passes information to host.
POST redeemVoucher
8. Host authorizes voucher redemption.
authorizeVoucher
9. Client passes response to end-client.
Redemption Authorized
10. End-client redeems voucher.
11. End-client reports redemption results.
Redemption Results
12. Client passes information to host.
POST commitVoucher
13. Host acknowledges redemption results.
commitVoucherAck
14. Client passes response to end-client.
Results Acknowledged

4.1.2 Voucher States

A voucher transaction will transition through a series of states while it is being processed by an end-client. Not all of these states are visible through the protocol. However, they are described here to provide guidance to implementers. The following diagrams identify the voucher transaction states and the permitted transitions that should be followed when implementing this specification.

NEW CHAPTER


Issuance Reported by End-Client
Issue Sent
Issuance Acknowledged by Host
Issue Acked


Redemption Requested by End-Client
Redeem Sent
Redemption Aborted by End-Client
Redemption Authorized by Host
Redemption Denied by Host
Redeem Auth
Redemption Aborted by End-Client
Redemption Completed by End-Client
Commit Sent
Final Result Acknowledged by Host
Commit Acked

4.1.3 Validation Identifiers

Validation identifiers are provided to the end-clients by the host. With each validation identifier, the host also provides a seed value.

- Validation identifiers MUST be 18-digit numeric values. Typically, the validation identifiers are printed on the vouchers in human-readable and bar-code form.
- Seed values MUST be constructed from 0 (zero) to 20 (twenty) UTF-8 encoded characters in the range U+0020 to U+007E (ASCII printable characters). The seed values are used to produce manual authentication identifiers. The manual authentication identifiers are also printed on the vouchers and can be used for offline validation of vouchers.

NEW CHAPTER

The configuration information provided by the host includes a series of properties that are used by the end-client to determine the number of validation identifiers to request from the host and to determine the frequency at which new validation identifiers should be requested.

- The `maxValIds` property indicates the maximum number of unused validation identifiers that should be stored by an end-client.
- The `minLevelValIds` property indicates the minimum number of unused validation identifiers stored by the end-client before additional validation identifiers should be requested.
- The `valIdListRefresh` property indicates the maximum time period that unused validation identifiers should be stored before new validation identifiers are requested.
- The `valIdListLife` property indicates the maximum time period before an end-client must stop using the validation identifiers.

Any time that the number of unused validation identifiers stored by an end-client drops below `minLevelValIds`, additional validation identifiers MUST be requested by the end-client. Similarly, if the validation identifiers have been stored for a period of time that exceeds the `valIdListRefresh` or `valIdListLife` limits, a new set of validation identifiers MUST be requested. In addition, when the end-client first starts communicating with the host, whenever the end-client resumes communications after an outage, and whenever voucher functionality is re-enabled after being disabled (that is, the `allowVoucherIssue` property of the voucher configuration is changed from false to true), a new set of validation identifiers MUST be requested.

When requesting validation identifiers, the number of validation identifiers requested by an end-client MUST NOT cause the number of unused validation identifiers stored by the end-client to exceed `maxValIds`.

If the unused validation identifiers have been stored for a period of time that exceeds `valIdListLife`, the validation identifiers MUST NOT be used to issue vouchers until the validation identifiers have been refreshed by the host.

4.1.4 Manual Authentication Identifiers

A manual authentication identifier MUST, if possible, be printed on every voucher for cashable or promotional credits produced by an end-client. And, when the `printNonCashOffLine` configuration property is set to true, a manual authentication identifier MUST, if possible, be printed on every voucher for non-cashable credits. The manual authentication identifier is derived from a 128-bit MD5 hash of the end-client identifier, validation identifier, seed value, and voucher amount.

Operational circumstances may prevent the end-client from printing manual authentication identifiers. For example, the operator might choose to configure the end-client to not print manual authentication identifiers. If the end-client cannot print manual authentication identifiers, the end-client MUST NOT print vouchers while offline. The end-client MUST behave as if the `printOffLine` configuration property was set to false.

The following procedure MUST be used to produce manual authentication identifiers.

1. Construct a 90-character string composed, from left to right, of:
    a. End-client identifier (`endClientId`); 32 8-bit ASCII characters (U+0020 to U+007E) padded right with zeros (U+0030).
    b. Validation identifier; 18 8-bit numeric ASCII characters (U+0030 to U+0039).
    c. Seed value; 20 8-bit UTF-8 encoded characters (U+0020 to U+007E) padded left with zeros (U+0030).

NEW CHAPTER

   d. Voucher amount represented in the minor unit of the base currency of the end-client with no punctuation or currency symbols; 20 8-bit numeric ASCII characters (U+0030 to U+0039) padded left with zeros (U+0030).

2. Convert all lower case ASCII characters (U+0061 to U+007A) in the composed string to upper case ASCII characters (U+0041 to U+005A).
3. Produce a 128-bit MD5 hash value using the 90-character composed string as input.
4. Produce the manual authentication identifier by casting the 128-bit hash value into a 32-character hexadecimal representation and converting all alphabetic characters to upper case (U+0041 to U+005A).

The G2S protocol includes additional information about producing manual authentication identifiers.

4.1.5 End-Client Identifier

The `endClientType` and `endClientId` properties uniquely identify a specific end-client within a gaming network. Client systems are responsible for identifying the end-client, determining the `endClientType` and `endClientId` for the end-client, and communicating the `endClientType` and `endClientId` to the host system. The host system is responsible for validating the `endClientType` and `endClientId` reported by client systems and accurately processing requests based on that information.

If the host determines that the `endClientType` and/or `endClientId` is invalid and, thus, the host is not permitted to process requests from the end-client, the host SHOULD respond with HTTP status code 409 "Conflict".

4.1.6 Player Identifier

When vouchers are issued, they may be associated with a player. The end-client is responsible for identifying the player, determining the player identifier for the player, and communicating the player identifier to the host system.

The host MUST NOT report errors related to invalid player information when processing voucher issuance requests. The host MUST make a best effort to accept voucher issuance requests even if the player information is invalid. The host MAY report errors related to invalid player information when processing voucher redemption requests (host exception code `6 Incorrect Player for Voucher`).

When identifying a player, the end-client MUST include either (a) the `playerId` for the player or (b) the `idReaderType` and `idNumber` of the ID presented by the player. Both sets of information MAY be included.

If both sets of information are included, the host SHOULD ignore the `idReaderType` and `idNumber` and only use the `playerId` to identify the player. If only one set of information is included, the host MAY include the other set of information in its response to the end-client. If neither set of information is included, the host MUST simply ignore the player information.

4.1.7 Configuration Identifier

The configuration identifier (`configurationId`) is used to identify a specific set of voucher configuration values. The host includes the configuration identifier with each set of voucher configuration values that it sends to an end-client. The host also includes the configuration identifier in each response that it sends to an end-client.

An end-client MUST compare the configuration identifiers received in responses to the configuration identifier for the current set of voucher configuration values being used by the end-client. If the configuration identifiers are different, the end-client MUST request a new set of voucher configuration values from the host.

The end-client MUST also request a new set of WAT configuration values when it first initiates communications with the host, whenever it resumes communications after an outage, and whenever it is re-enabled after being disabled.

4.1.8 Transaction Identifier

Each voucher transaction (issuance or redemption) is assigned a transaction identifier (`transactionId`). The end-client is responsible for assigning the `transactionId` to the transaction.

The `transactionId` MUST uniquely identify a specific voucher transaction for the end-client. The combination of `endClientType`, `endClientId`, and `transactionId` MUST be unique. Provided that a `transactionId` is unique to the end-client, the end-client MAY use whatever method it determines appropriate to assign the `transactionId`.

The host system MUST use the `transactionId` to detect duplicate transactions coming from an end-client, as well as updates to transactions. Since the `transactionId` is only unique to the end-client, the host system MUST use the combination of `endClientType`, `endClientId`, and `transactionId` to uniquely identify individual transactions.

The `transactionId` assigned to a voucher transaction is selected by the end-client when the transaction is first initiated. The end-client is responsible for properly recording the `transactionId` and including it in any subsequent requests related to the transaction. The host system is responsible for properly recording the `transactionId` when the transaction is first reported and, subsequently, accurately detecting duplicates and making appropriate updates based on the `transactionId` provided by the end-client.

4.1.9 Host Exception Code

The host exception code (`hostException`) is used by the host to report exceptions that may occur while processing requests. A non-zero value indicates that an exception occurred and, typically, that the request was not successful at the host. A value of 0 (zero) indicates that no exception occurred and that the request was successful at the host. Individual resources may contain additional rules for reporting and handling specific exception codes.

4.1.10 End-Client Exception Code

The end-client exception code (`endClientException`) is used by the end-client to report exceptions that may occur while processing requests. A non-zero value indicates that an exception occurred and, typically, that the request was not successful at the end-client. A value of 0 (zero) indicates that no exception occurred and that the request was successful at the end-client. Individual resources may contain additional rules for reporting and handling specific exception codes.

4.1.11 Employee Authorizations

Certain voucher redemptions may require authorization by an employee of the gaming operation. For example, an employee authorization may be required when redeeming large-win vouchers. The number of employees who must authorize a redemption is determined by the host and is communicated to the end-client through `requiredAuth` sub-objects of the `voucherStatus` resource. The number of employees may vary depending on the value of a voucher, the type of voucher, or other factors. For example, a host may require one employee authorization for redemptions of large-win vouchers up to $10,000 and two authorizations when over $10,000. Employee authorizations are reported in `employeeAuth` sub-objects of the `redeemVoucher` resource.

When multiple authorizations are required, different types of employees may be required for the different authorizations. For example, when two authorizations are required, a supervisor and a manager may be required to authorize the redemption. The types of employee required to authorize the redemption of a voucher are communicated through job codes (`jobCode`). Employees may be assigned multiple job codes to reflect the full range of authorizations that they may make. For example, an employee might be assigned a "supervisor" job code and a "manager" job code; this would allow the employee to provide supervisor and manager authorizations. Another employee might only be assigned a "cashier" job code.

Along with the list of job codes that are required to authorize a redemption, the host MAY also specify authorization codes (`authCode`). Authorization codes allow the host to be more specific about the types of authorizations that are required. For example, rather than just specifying that a supervisor and manager are required to authorize a redemption, the host can specify that the supervisor must provide the first authorization and that the manager must provide the second.

Some authorization codes are mandatory — that is, when the host specifies the authorization code, the end-client MUST include the authorization code in the voucher redemption request along with the employee identifier of the authorizing employee. For example, authorization code `SSI_authLine1` is mandatory; when authorization code `SSI_authLine1` is specified by the host, the end-client must include authorization code `SSI_authLine1` in its redemption request along with the employee identifier of the authorizing employee.

Other authorization codes are optional — that is, when the host specifies the authorization code, the end-client MAY take the action associated with the authorization code. If the action is taken, the end-client MUST include the authorization code with the employee identifier of the authorizing employee in its redemption request. For example, authorization code `SSI_changeAmount` is optional; when authorization code `SSI_changeAmount` is specified by the host, the end-client may change the value of the voucher; if the value of the voucher is changed, the end-client must include authorization code `SSI_changeAmount` in its redemption request along with the employee identifier of the authorizing employee.

The host MAY specify more than one job code for each authorization code. To qualify to perform a specific authorization, an employee must have at least one of the job codes specified for the authorization code. For example, if the host specifies that a supervisor or a manager must perform a specific authorization, an employee must have a supervisor or manager job code to perform the authorization. However, the end-client MUST only include one employee authorization record per authorization code. For example, if the host specifies that a supervisor or a manager must perform a specific authorization, the end-client must only include one employee authorization record; that record must identify the supervisor or manager who authorized the redemption.

See Table 6.3,t_authCodes Enumerations for a list of mandatory and optional authorization codes.

Thus, there are three types of authorizations that can be specified by the host. The processing requirements are different for each type.

- Generic – No authorization code is specified by the host. The end-client MUST include an employee authorization record in its redemption request for each job code specified by the host.
- Mandatory – A mandatory authorization code is specified by the host. The end-client MUST include one and only one employee authorization record for each mandatory authorization code specified by the host regardless of the number of job codes specified for the mandatory authorization code.
- Optional – An optional authorization code is specified by the host. The end-client MUST include one and only one employee authorization record for each optional authorization code specified by the host regardless of the number of job codes specified for the mandatory authorization code if the specific action associated with the optional authorization code is taken. If the action is not taken, the end-client MUST NOT include an employee authorization record for the optional authorization code.

NEW CHAPTER

4.1.12 Optional Properties

In this specification, properties of objects and URIs are designated as either `use: required` or `use: optional`.

When designated as `use: required`, the property MUST be included in the object or URI. If one or more `use: required` properties of an object or URI is omitted, the object or URI MUST be considered syntactically and semantically incorrect. In such cases, if the incorrect object is in a request, the recipient MUST respond with exception code `98 Syntax or Semantic Error`; if the incorrect object is in a response, the recipient SHOULD log the error and attempt to notify the system operator; if the URI is incorrect, the recipient MUST respond with HTTP status code 409 “Conflict”.

When designated as `use: optional`, the property MAY be omitted. If omitted, the recipient of the object or URI MUST use the specified default value for the property as the value of the property. If no default value is specified, the recipient MUST use the null (unknown) value as the value of the property. Individual resources may contain additional rules for the handling of `use: optional` properties.

NEW CHAPTER

4.2 GET voucherConfiguration Resource

This resource is used to request new voucher configuration settings from the host.

Table 4.1 GET voucherConfiguration Resource

HTTP Method	`GET`
Pathname	`/ssi/[ver]/voucherConfiguration`
Request Content-Type	`application/json; charset=utf-8`
Request Content	None.
Response Content - Type	`application/json; charset=utf-8`
Response Content	`voucherConfiguration` Object.

4.2.1 GET voucherConfiguration Properties

The following table identifies the properties for the `voucherConfiguration` resource when the HTTP GET verb is used. The properties are appended to the resource URI in the query component of the HTTP request.

An end-client MUST request new voucher configuration settings when it first starts communicating with the host, whenever it resumes communications after an outage, whenever the voucher functionality is re-enabled after being disabled, and whenever a mismatch is detected between the configuration identifier reported by the host and the current configuration identifier being used by the end-client. The end-client MAY request the current voucher configuration settings at other times as well.

In such cases, the end-client MUST retry the request at the frequency specified in the `timeToLive` configuration property (or, 30 seconds if no voucher configuration setting were ever received) until new configuration settings are received. Until new configuration settings are received, the end-client MUST NOT generate any other voucher-related requests.

If the request is successful — that is, the host is providing the voucher configuration to the end-client in the response — the `hostException` property of the response MUST be set to 0 (zero). Otherwise, the `hostException` property MUST be set to an appropriate non-zero value; all other optional properties of the response MUST be omitted. If the request is not successful and an appropriate value for the `configurationId` property cannot be determined by the host, the host MUST set the `configurationId` property to 0 (zero).

- If the end-client is unknown or invalid, the host MUST set the `hostException` property to `97 Unknown` or `Invalid End-Client`.
- If the voucher configuration is not available for the end-client, the host MUST set the `hostException` property to `20 Voucher Configuration Not Available`.

Table 4.2 GET voucherConfiguration Properties

Property	Restrictions	Description

`endClientType`	`type: t_clientTypes`
	`use: required`
		End-client type.
		Example, "SSI_kiosk".

NEW CHAPTER

Table 4.2 GET voucherConfiguration Properties (Continued)

Property	Restrictions	Description
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".

4.2.2 voucherConfiguration Object

The following table identifies the properties of the `voucherConfiguration` Object. Additional properties MAY be included in the object.

- The `currencyCode` property identifies the currency in which voucher transactions are denominated. If the currency specified by the host does not match the currency in which the end-client operates, the end-client MUST NOT initiate voucher issuance or redemption transactions with the host.
- The `timeToLive` property indicates the minimum amount of time that the end-client should wait before retrying a request. To retry a request, it may be necessary to terminate and then re-establish the HTTP connection to the host.
- The `noAckTimer` property is used to determine whether voucher processing services are offline. When the end-client issues a voucher, the end-client MUST start a timer using the value of the `noAckTimer` property. If the timer expires before the host acknowledges issuance of the voucher, the end-client MUST consider voucher processing services to be offline. Once all issued vouchers have been acknowledged by the host, the end-client MUST consider voucher processing services to be online again.

Table 4.3 voucherConfiguration Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
currencyCode	type:t_currencyCode use: optional default: XXX	Currency Code; the currency in which transactions are denominated. Example, "EUR".
timeToLive	type: t_milliseconds use: optional default: 30000	Time-to-live value for voucher-related requests generated by the end-client. Example, "30000".
combineCashableOut	type: boolean use: optional default: true	Indicates whether promotional credits are converted to cashable credits when issuing vouchers. Example, "true".

Table 4.3 voucherConfiguration Object Properties (Continued)

Property	Restrictions	Description
allowNonCashOut	type: boolean use: optional default: false	Indicates whether the end-client is allowed to issue vouchers for non-cashable credits. Example, "false".
maxValIds	type: numeric mode: integer use: optional default: 15 minimum: 1	Maximum number of validation identifiers stored by the end-client. Example, "15".
minLevelValIds	type: numeric mode: integer use: optional default: 10 minimum: 0	Minimum number of validation identifiers stored by the end-client before additional validation identifiers are requested. Example, "10".
validListRefresh	type: t_milliseconds use: optional default: 43200000	Maximum time period after validation identifiers have been refreshed before new validation identifiers are requested. Example, "43200000".
validListLife	type: t_milliseconds use: optional default: 86400000	Maximum time period after validation identifiers have been refreshed before the end-client must stop using the validation identifiers. Example, "86400000".
voucherHoldTime	type: t_milliseconds use: optional default: 15000	Maximum time period that the end-client should wait for a host authorization before returning a voucher. Example, "15000".
printOffLine	type: boolean use: optional default: true	Indicates whether the end-client is allowed to issue vouchers while communications to the host are offline. Example, "true".
expireCashPromo	type: numeric mode: integer use: optional default: 30 minimum: 0	Number of days before vouchers for cashable and promotional credits expire. Example, "30".
printExpCashPromo	type: boolean use: optional default: true	Indicates whether expiration dates should be printed on vouchers for cashable and promotional credits. Example, "true".

NEW CHAPTER

Table 4.3 voucherConfiguration Object Properties (Continued)

Property	Restrictions	Description
expireNonCash	type: numeric mode: integer use: optional default: 30 minimum: 0	Number of days before vouchers for non-cashable credits expire. Example, "30".
printExpNonCash	type: boolean use: optional default: true	Indicates whether expiration dates should be printed on vouchers for non-cashable credits. Example, "true".
propName	type: t_voucherTitle40 use: optional default: <empty>	Name of the property. Example, "ABC Casino".
propLine1	type: t_voucherTitle40 use: optional default: <empty>	First address line for the property. Example, "1 Casino Way".
propLine2	type: t_voucherTitle40 use: optional default: <empty>	Second address line for the property. Example, "Anywhere, USA".
titleCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for cashable credits. Example, "CASHOUT VOUCHER".
titlePromo	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for promotional credits; if <empty>, use titleCash. Example, "CASHOUT VOUCHER".
titleNonCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for non-cashable credits. Example, "PLAYABLE ONLY".
titleLargeWin	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for wins greater than the large win limit for the client. Example "JACKPOT VOUCHER".
titleShortPay	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for issued as a result of short pays. Example "SHORT PAY".
titleBonusCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for cashable credits resulting from external bonus awards. Example, "CASHOUT VOUCHER".
titleBonusPromo	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for promotional credits resulting from external bonus awards; if <empty>, use titleBonusCash. Example, "CASHOUT VOUCHER".

Table 4.3 voucherConfiguration Object Properties (Continued)

Property	Restrictions	Description
titleBonusNonCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for non-cashable credits resulting from external bonus awards. Example, "PLAYABLE ONLY".
titleWatCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for cashable credits resulting from wagering account transfers. Example, "CASHOUT VOUCHER".
titleWatPromo	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for promotional credits resulting from wagering account transfers; if <empty>, use titleWatCash. Example, "CASHOUT VOUCHER".
titleWatNonCash	type: t_voucherTitle16 use: optional default: <empty>	Title printed on vouchers for non-cashable credits resulting from wagering account transfers. Example, "PLAYABLE ONLY".
allowVoucherIssue	type: boolean use: optional default: true	Indicates whether the end-client should request validation identifiers, thus, enabling voucher issuance functionality. Example, "true".
allowVoucherRedeem	type: boolean use: optional default: true	Indicates whether the end-client should enable voucher redemption functionality. Example, "true".
maxOnLinePayOut	type: t_millicents use: optional default: 0	Maximum amount that can be paid by voucher while communications are not offline; 0 (zero) indicates that there is no limit. Example, "0".
maxOffLinePayOut	type: t_millicents use: optional default: 0	Maximum amount that can be paid by voucher while communications are offline; 0 (zero) indicates that there is no limit. Example, "1000000000".
printNonCashOffLine	type: boolean use: optional default: false	Indicates whether vouchers for non-cashable credits can be issued while communications are lost; both printOffLine and allowNonCashOut must also be set to true for vouchers for non-cashable credits to be printed while communications are lost. Example, "false".

NEW CHAPTER

Table 4.3 voucherConfiguration Object Properties (Continued)

Property	Restrictions	Description
noAckTimer	type: t_milliseconds use: optional default: 15000	The maximum time between when a voucher is issued and when it is acknowledged before the validation system is declared offline. Example, "15000".
hostException	type: t_voucherHostExceptions use: optional default: 0	Host exception code; 0 (zero) if request was successful. Example, "0".

4.2.3 GET voucherConfiguration Example — Successful

The following example demonstrates the construction of a successful GET `voucherConfiguration` request and a response containing a `voucherConfiguration` object. In practice, additional HTTP headers may be included in the message.

Request:

```
GET /ssi/1.1/voucherConfiguration?endClientType=SSI_kiosk&endClientId=ABC_123 HTTP/1.1
Accept: application/json
Accept-Charset: utf-8
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 1014
Content-Type: application/json; charset=utf-8

{
   "endClientType": "SSI_kiosk",
   "endClientId": "ABC_123",
   "configurationId": 1235813,
   "timeToLive": 15000,
   "combineCashableOut": true,
   "allowNonCashOut": false,
   "maxValIds": 15,
   "minLevelValIds": 10,
   "valIdListRefresh": 43200000,
   "valIdListLife": 86400000,
   "voucherHoldTime": 15000,
   "printOffLine": true,
   "expireCashPromo": 30,
   "printExpCashPromo": true,
   "expireNonCash": 30,
   "printExpNonCash": true,
   "propName": "ABC Casino",
   "propLine1": "1 Casino Way",
   "propLine2": "Anywhere, USA",
   "titleCash": "CASHOUT VOUCHER",
   "titlePromo": "CASHOUT VOUCHER",
   "titleNonCash": "PLAYABLE ONLY",
   "titleLargeWin": "JACKPOT VOUCHER",
    "titleShortPay": "SHORT PAY",
    "titleBonusCash": "CASHOUT VOUCHER",
    "titleBonusPromo": "CASHOUT VOUCHER",
    "titleBonusNonCash": "PLAYABLE ONLY",
    "titleWatCash": "CASHOUT VOUCHER",
    "titleWatPromo": "CASHOUT VOUCHER",
    "titleWatNonCash": "PLAYABLE ONLY",
    "allowVoucherIssue": true,
    "allowVoucherRedeem": true,
    "maxOnLinePayOut": 0,
    "maxOffLinePayOut": 1000000000,
    "printNonCashOffLine": false,
    "noAckTimer": 15000,
    "hostException": 0
}
```

4.2.4 Get voucherConfiguration Example — Not Successful

The following example demonstrates the construction of an unsuccessful GET `voucherConfiguration` request and a response containing a `voucherConfiguration` object. In practice, additional HTTP headers may be included in the message.

Request:

```
GET /ssi/1.1/voucherConfiguration?endClientType=SSI_kiosk&endClientId=ABC_123 HTTP/1.1
Accept: application/json
Accept-Charset: utf-8
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 75
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 0,
    "hostException": 20
}
```

NEW CHAPTER

4.3 GET validationResource

This resource is used to request a new set of validation identifiers from the host.

All vouchers that have been issued by an end-client MUST be acknowledged by the host before any requests for new validation identifiers are generated by the end-client.

Provided that all vouchers issued by the end-client have been acknowledged, the end-client MUST request new validation identifiers under the following circumstances:

- When the number of validation identifiers stored for the end-client falls below the `minLevelValIds` limit,
- When the `valIdListRefresh` or `valIdListLife` time period expires, or
- When the voucher functionality is re-enabled after being disabled (that is, the `allowVoucherIssue` configuration property is changed from false to true).

Once the end-client has determined that the validation identifiers need to be refreshed, the end-client MUST retry the request at the frequency specified in the `timeToLive` configuration property until new validation identifiers are received.

If the `valIdListLife` time period has expired or the end-client has never received any validation identifiers, the end-client MUST set the `valIdListExpired` property of the request to true; otherwise, the `valIdListExpired` property MUST be set to false.

Provided that the `valIdListLife` time period has not expired, the end-client may continue to issue vouchers until all available validation identifiers have been consumed. However, after the voucher functionality is re-enabled after being disabled (that is, the `allowVoucherIssue` configuration property is changed from false to true), the end-client MUST NOT issue any vouchers until the validation identifiers have been refreshed – that is, the end-client MUST treat the existing validation identifiers as if the `valIdListLife` time period had expired.

The `numValidationIds` property MUST be set to the difference between the value of the `maxValIds` configuration property and the number of unused validation identifiers remaining at the end-client, but not less than 0 (zero).

The `validationListId` property MUST be set to the value of the `validationListId` property received with the last set of validation identifiers (if no validation identifiers were ever received, MUST be set to 0 (zero)).

The end-client MUST NOT request validation identifiers and, thus, MUST NOT issue vouchers if the `allowVoucherIssue` configuration property is set to false.

If the request is successful — that is, the host is providing the validation identifiers to the end-client in the response — the `hostException` property of the response MUST be set to 0 (zero). Otherwise, the `hostException` property MUST be set to an appropriate non-zero value; all other optional properties of the response MUST be omitted; the `configurationId` and `validationListId` properties MUST be set to the corresponding values received in the request.

- If the end-client is unknown or invalid, the host MUST set the `hostException` property to `97 Unknown or Invalid End-Client`.
- If the voucher configuration is not correct, the host MUST set the `hostException` property to `21 Incorrect Voucher Configuration`.

Table 4.4 GET validationIdList Resource

HTTP Method	`GET`

Table 4.4 GET validationIdList Resource

Pathname	/ssi/[ver]/validationIdList
Request Content-Type	application/json; charset=utf-8
Request Content	None.
Response Content - Type	application/json; charset=utf-8
Response Content	validationIdList Object.

4.3.1 Get validationIdList Properties

The following table identifies the properties for the validationIdList resource when the HTTP GET verb is used. The properties are appended to the resource URI in the query component of the HTTP request.

Table 4.5 GET validationList Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813"
validationListId	type: t_validationListId use: required	Validation List Identifier; the validationListId received in the last validationIdList object received by the client; set to 0 (zero) if no such object has been received. Example, “13471118”.
numValidationIds	type: numeric mode: integer use: required minimum: 0	The number of validation identifiers required to restore the number of available validation identifiers to the maxValIds level, but not less than 0 (zero). Example, "5".
validListExpired	type: boolean use: required	Indicates whether the valIdListLife time period is considered to have expired; set to true if no validation identifiers have ever been received by the end-client. Example, "false".

NEW CHAPTER

4.3.2 validationIdList Object

The following table identifies the properties of the validationIdList Object. Additional properties MAY be included in the object.

The host can use the deleteCurrent property to indicate that all unused validation identifiers should be discarded before adding the new validation identifiers.

If the deleteCurrent property is set to false, the host MUST include the number of validation identifiers specified in the numValidationIds property of the request. If the deleteCurrent property is set to true, the host MUST include the number of validation identifiers specified in the maxValIds configuration parameter.

If the deleteCurrent property is set to true, all unused validation identifiers MUST be discarded by the end-client. If the deleteCurrent property is set to false, all unused validation identifiers MUST be retained by the end-client.

Validation identifiers MUST be consumed by the end-client in the order provided. Any new validation identifiers MUST be consumed after any validation identifiers retained by the end-client. If any validation identifiers are duplicates of those retained by the end-client, the ordering of the validation identifiers MUST NOT be changed, however, the seed values associated with the validation identifiers MUST be updated.

After successfully recording a new set of validation identifiers, the end-client MUST restart any timers associated with the valIdsListRefresh and valIdsListLife configuration properties.

If any of the new validation identifiers or seed values cannot be used – for example, a validation identifier includes non-numeric characters – the entire new set of validation identifiers MUST NOT be used. In such cases, the end-client SHOULD, if possible, use its local error reporting mechanisms to try to bring the problem to the operator’s attention.

Table 4.6 validationIdList Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
validationListId	type: t_validationListId use: required	Validation List Identifier; host-assigned identifier for the set of validation identifiers. Example, "13471118".
deleteCurrent	type: boolean use: optional default: false	Indicates whether all remaining validation identifiers stored by the client should be discarded. Example, "false".
validationIdArray	type: array use: optional minItems: 0 maxItems: ∞	An array of validationId objects. See Table 4.7,validationId Object Properties for details.

NEW CHAPTER

Table 4.6 validationIdList Object Properties

Property	Restrictions	Description
`hostException`	type: `t_voucherHostExceptions` use: optional default: 0	Host exception code; 0 (zero) if request was successful. Example, “0”.

Table 4.7 validationId Object Properties

Property	Restrictions	Description
`validationId`	type: `t_validationId` use: required	Validation Identifier. Example, "012345678901234567".
`validationSeed`	type: `t_validationSeed` use: optional default: <empty>	Manual authentication seed value. Example, "1A2B3C4D5E6F7081".

4.3.3 GET validationIdList Example — Successful

The following example demonstrates the construction of a successful GET `validationIdList` request and a response containing a `validationIdList` object. In practice, additional HTTP headers may be included in the message.

Request:

```
GET /ssi/1.1/validationIdList?endClientType=SSI_kiosk&endClientId=ABC_123
    &configurationId=1235813&validationListId=1347118
    &numValidationIds=5&valIdListExpired=false HTTP/1.1
Accept: application/json
Accept-Charset: utf-8
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 537
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "validationListId": 13471119,
    "deleteCurrent": false,
    "validationIdArray": [
        {
            "validationId": "012345678901234567",
            "validationSeed": "1A2B3C4D5E6F7081"
        },
        {
            "validationId": "012345678901234568",
            "validationSeed": "1A2B3C4D5E6F7081"
        },
        {
            "validationId": "012345678901234569",
            "validationSeed": "1A2B3C4D5E6F7081"
        },
        {
            "validationId": "012345678901234570",
            "validationSeed": "1A2B3C4D5E6F7081"
        },
        {
            "validationId": "012345678901234571",
            "validationSeed": "1A2B3C4D5E6F7081"
        }
    ]
}
```

4.3.4 GET validationIdList Example — Not Successful

The following example demonstrates the construction of an unsuccessful GET `validationIdList` request and a response containing a `validationIdList` object. In practice, additional HTTP headers may be included in the message.

Request:

```
GET /ssi/1.1/validationIdList?endClientType=SSI_kiosk&endClientId=ABC_123
    &configurationId=1235813&validationListId=1347118
    &numValidationIds=5&valIdListExpired=false HTTP/1.1
Accept: application/json
Accept-Charset: utf-8
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 131
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "validationListId": 13471119,
    "hostException": 21
}
```

NEW CHAPTER

4.4 POST issueVoucher Resource

This resource is used to report to the host that a new voucher has been issued.

Table 4.8 POST issueVoucher Resource

HTTP Method	`POST`
Pathname	`/ssi/[ver]/issueVoucher`
Request Content-Type	`application/json; charset=utf-8`
Request Content	`issueVoucher` Object.
Response Content - Type	`application/json; charset=utf-8`
Response Content	`issueVoucherAck` Object.

4.4.1 issueVoucher Object

The following table identifies the properties of the `issueVoucher` Object. Additional properties MAY be included in the object.

The end-client should report that a voucher has been issued as soon as the end-client is irreversibly committed to the voucher issuance operation and the associated credits have been removed from the credit meter. The end-client should not wait until the final results of the print operation are known. Waiting for the final results of the print operation could cause significant delays in reporting that the voucher issuance operation had taken place. Presentation errors MAY be reported by setting the `endClientException` property to 1 (one). However, reporting such errors SHOULD NOT delay the reporting of the voucher issuance operation.

The end-client MUST retry the `issueVoucher` request at the frequency set in the `timeToLive` configuration property until the voucher issuance is acknowledged by the host.

When issuing vouchers for non-cashable credits, the following rules MUST be applied:

- If the `allowNonCashOut` configuration property is set to true:
    - If there is no expiration associated with the non-cashable credits, the end-client MUST produce the voucher for the non-cashable credits and the `expireNonCash` configuration property MUST be used to determine the expiration period for the voucher (not the expiration date/time for the non-cashable credits).
    - If there is an expiration associated with the non-cashable credits and the current date/time is the same as or prior to that expiration, the end-client MUST produce the voucher for the non-cashable credits.
    - If there is an expiration associated with the non-cashable credits and the current date/time is after that expiration, the end-client MUST NOT produce a voucher for the non-cashable credits.
- If the `allowNonCashOut` configuration property is set to false, the end-client MUST NOT produce a voucher for the non-cashable credits.

When the `combineCashableOut` configuration property is set to true, the end-client MUST convert any promotional credits to cashable credits when issuing a voucher — a single combined voucher for cashable credits MUST be issued. When the `combineCashableOut` configuration attribute is set to false, the end-client

MUST NOT convert promotional credits to cashable credits when issuing a voucher — when necessary, separate vouchers for cashable and promotional credits MUST be issued.

If the voucher was issued as a partial payment because the end-client could not fully redeem another voucher, the end-client MUST set the shortPay property to true; otherwise, the shortPay property MUST be set to false. For example, if a disbursement error occurs at a redemption kiosk and, as a result, the kiosk only partially redeems a voucher, issuing a new voucher for the remaining balance, the kiosk must set the shortPay property for the new voucher to true.

An issueVoucher request is considered logically equivalent to a previous issueVoucher request if the host detects that the transactionId associated with the request was reported in a previous issueVoucher request for the same end-client. In such cases, the host MUST generate a logically equivalent issueVoucherAck response.

The host SHOULD also verify that the validationId has not been reported in a previous issueVoucher request. If the validationId has been previously reported, but with a different transactionId, the host SHOULD, if possible, use its local error reporting mechanisms to try to bring the problem to the operator's attention.

The host MUST make a best effort to acknowledge an issueVoucher request. Failure to acknowledge an issueVoucher request will cause the end-client to retry the issueVoucher request indefinitely and may lead to a loss of voucher-related functionality. Host exception 21 Incorrect Voucher Configuration MUST NOT be reported in response an issueVoucher request. If the request is not successful, the configurationId, transactionId, and validationId properties MUST be set to the corresponding values received in the request.

Table 4.9 issueVoucher Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
transactionId	type: t_transactionId use: required	Transaction identifier. Example, "14591423".
idReaderType	type: t_idReaderTypes use: optional default: SSI_none	ID reader type. Example, "SSI_magCard".
idNumber	type: t_idNumber use: optional default: <empty>	ID number. Example, "09900101977".
playerId	type: t_playerId use: optional default: <empty>	Player account identifier. Example, "00101977".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".

NEW CHAPTER

Table 4.9 issueVoucher Object Properties

Property	Restrictions	Description
voucherAmt	type: t_millicents use: required	Voucher amount. Example, "12345000".
creditType	type: t_creditTypes use: required	Credit Type. Example, "SSI_cashable".
voucherSource	type: t_voucherSources use: required	Voucher source; set to SSI_endClient. Example, "SSI_endClient".
largeWin	type: boolean use: required	Indicates whether the voucher was issued because the amount won exceeded the end-client's large win limit. Example, "false".
shortPay	type: boolean use: required	Indicates whether the voucher was issued because the end-client could not fully redeem another voucher. Example, "false".
voucherSequence	type: numeric mode: integer use: required minimum: 0	The issuing end-client's internal voucher sequence number; printed on the voucher. Example, "123".
expireCredits	type: boolean use: required	Indicates whether non-cashable credits have an associated expiration date/time; ignored when creditType is not set to SSI_nonCashable. Example, "false".
expireDateTime	type: string use: required format: date-time	Expiration date/time associated with non-cashable credits; ignored when creditType is not set to SSI_nonCashable or expireCredits is set to false. Example, "2001-01-01T00:00:00-00:00".
transferAmt	type: t_millicents use: required	Transfer amount; in most cases, set to voucherAmt. MAY be set to 0 (zero) if an exception occurred while generating the voucher and no funds were transferred from the end-client. Example, "12345000".
transferDateTime	type: string use: required format: date-time	Date/time that the transaction took place. Example, "2016-03-31T17:11:28-05:00".

Table 4.9 issueVoucher Object Properties

Property	Restrictions	Description
expireDays	type: numeric mode: integer use: required minimum: 0	Number of days before the voucher expires; ignored if expireCredits is set to true; -1 (negative one) indicates that there is no expiration period. Example, "30".
endClientAction	type: t_voucherClientActions use: required	Action taken by the end-client; set to SSI_issued. Example, "SSI_issued".
endClientException	type: t_voucherClientExceptions use: required	End-client exception code. Example, "0".

4.4.2 issueVoucherAck Object

The following table identifies the properties of the issueVoucherAck Object. Additional properties MAY be included in the object.

Table 4.10 issueVoucherAck Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
transactionId	type: t_transactionId use: required	Transaction identifier. Example, "14591423".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".
hostException	type: t_voucherHostExceptions use: optional default: 0	Host exception code; 0 (zero) if request was successful. Example, "0".

NEW CHAPTER

4.4.3 POST issueVoucher Example — Successful

The following example demonstrates the construction of a successful POST `issueVoucher` request containing a `issueVoucher` object and a response containing a `issueVoucherAck` object. In practice, additional HTTP headers may be included in the message.

```
Request:
POST /ssi/1.1/issueVoucher HTTP/1.1
Content-Length: 561
Content-Type: application/json; charset=utf-8
Accept: application/json
Accept-Charset: utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591423,
    "idReaderType": "SSI_magCard",
    "idNumber": "09900101977",
    "playerId": "00101977",
    "validationId": "012345678901234567",
    "voucherAmt": 12345000,
    "creditType": "SSI_cashable",
    "voucherSource": "SSI_endClient",
    "largeWin": false,
    "shortPay": false,
    "voucherSequence": 123,
    "expireCredits": false,
    "expireDateTime": "",
    "transferAmt": 12345000,
    "transferDateTime": "2016-03-31T17:11:28-05:00",
    "expireDays": 30,
    "endClientAction": "SSI_issued",
    "endClientException": 0
}
```

```
Response:
HTTP/1.1 200 OK
Content-Length: 108
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591423,
    "validationId": "012345678901234567"
}
```

NEW CHAPTER

4.5 GET voucherStatus Resource

This resource is used to request the current status of a voucher from the host. Typically, the resource is only used when the host requires employee authorizations for certain types of vouchers. The list of required employee authorizations for a voucher is included with the voucher status information. The resource may be used for other purposes as well.

Table 4.11 GET voucherStatus Resource

HTTP Method	GET
Pathname	/ssi/[ver]/voucherStatus
Request Content-Type	application/json; charset=utf-8
Request Content	None.
Response Content-Type	application/json; charset=utf-8
Response Content	`voucherStatus` Object.

4.5.1 GET voucherStatus Properties

The following table identifies the properties for the `voucherStatus` resource when the HTTP GET verb is used. The properties are appended to the resource URI in the query component of the HTTP request.

If the request is successful – that is, the host is providing voucher status information to the end-client in the response – the `hostException` property of the response MUST be set to 0 (zero). Otherwise, the `hostException` property MUST be set to an appropriate non-zero value; all other optional properties of the response MUST be omitted; the `configurationId` and `validationId` properties MUST be set to the corresponding values received in the request.

- If the end-client is unknown or invalid, the host MUST set the hostException property to `97 Unknown or Invalid End-Client`.
- If the voucher configuration is not correct, the host MUST set the `hostException` property to `21 Incorrect Voucher Configuration`.
- If the validationId is unknown or invalid, the host MUST set the `hostException` property to `4 Voucher Not Found`.

Table 4.12 GET voucherStatus Properties

Property	Restrictions	Description

`endClientType`	`type: t_clientTypes`
	`use: required`
		End-client type.
		Example, "SSI_kiosk".

`endClientId`	`type: t_clientId`
	`use: required`
		End-client identifier.
		Example, "ABC_123".

`configurationId`	`type: t_configurationId`
	`use: required`
		Configuration identifier.
		Example, "1235813".

NEW CHAPTER

Table 4.12 GET voucherStatus Properties

Property	Restrictions	Description
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".

4.5.2 voucherStatus Object

The following table identifies the properties of the voucherStatus Object. Additional properties MAY be included in the object.

- The voucherStatus property identifies the current state of the voucher. Vouchers in the SSI_issueAcked state are eligible for redemption.

Table 4.13 voucherStatus Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".
voucherStatus	type: t_voucherStates use: optional default: SSI_issueAcked	Voucher status. Example, "SSI_issueAcked".
voucherAmt	type: t_millicents use: optional default: 0	Voucher amount. Example, "12345000".
creditType	type: t_creditTypes use: optional default: SSI_cashable	Credit Type. Example, "SSI_cashable".
voucherSource	type: t_voucherSources use: optional default: SSI_endClient	Voucher source; set to SSI_endClient. Example, "SSI_endClient".
largeWin	type: boolean use: optional default: false	Indicates whether the voucher was issued because the amount won exceeded the end-client's large win limit. Example, "false'.

Table 4.13 voucherStatus Object Properties

Property	Restrictions	Description
shortPay	type: boolean use: optional default: false	Indicates whether the voucher was issued because the end-client could not fully redeem another voucher. Example, "false".
voucherSequence	type: numeric mode: integer use: optional default: 0 minimum: 0	The issuing end-client's internal voucher sequence number; printed on the voucher. Example, "123".
expireCredits	type: boolean use: optional default: false	Indicates whether non-cashable credits have an associated expiration date/time; ignored when creditType is not set to SSI_nonCashable. Example, "false".
expireDateTime	type: string use: optional default: <empty> format: date-time	Expiration date/time associated with non-cashable credits; ignored when creditType is not set to SSI_nonCashable or expireCredits is set to false. Example, "2001-01-01T00:00:00-00:00".
requiredAuthArray	type: array use: optional minItems: 0 maxItems: ∞	An array of requiredAuth objects. See Table 4.14,requiredAuth Object Properties for details.
hostException	type: t_voucherHostExceptions use: optional default: 0	Host exception code; 0 (zero) if request was successful. Example, "0".

Table 4.14 requiredAuth Object Properties

Property	Restrictions	Description
authCode	type: t_authCodes use: optional default: <empty>	Authorization code. Example, "SSI_authLine1".
authName	type: t_voucherTitle40 use: optional default: <empty>	Authorization title. Example, "Authorization 1".
jobCode	type: t_jobCode use: required	Job code. For example, "manager".
jobName	type: t_voucherTitle40 use: optional default: <empty>	Job title. For example, "Manager or Above".

NEW CHAPTER

4.5.3 GET voucherStatus Example — Successful

The following example demonstrates the construction of a successful GET `voucherStatus` request and a response containing a `voucherStatus` object. In practice, additional HTTP headers may be included in the message.

In this example, the end-client is required to provide a generic employee authorization by an "attendant". If the value of the voucher is changed, an optional employee authorization by a "supervisor" or "manager" is also required.

```
Request:
GET /ssi/1.1/voucherStatus?endClientType=SSI_kiosk&endClientId=ABC_123
    &configurationId=1235813&validationId=012345678901234567 HTTP/1.1
Accept: application/json
Accept-Charset: utf-8


Response:
HTTP/1.1 200 OK
Content-Length: 686
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "validationId": "012345678901234567",
    "voucherStatus": "SSI_issueAcked",
    "voucherAmt": 12345000,
    "creditType": "SSI_cashable",
    "voucherSource": "SSI_endClient",
    "largeWin": false,
    "shortPay": false,
    "voucherSequence": 123,
    "expireCredits": false,
    "expireDateTime": "",
    "requiredAuthArray": [
        {
            "authCode": "",
            "authName": "",
            "jobCode": "attendant",
            "jobName": "Attendant"
        },
        {
            "authCode": "SSI_changeAmt",
            "authName": "Change Voucher Amount",
            "jobCode": "supervisor",
            "jobName": "Supervisor or Above"
        },
        {
            "authCode": "SSI_changeAmt",
            "authName": "Change Voucher Amount",
            "jobCode": "manager",
            "jobName": "Manager or Above"
        }
    ]
}
```

NEW CHAPTER

4.5.4 GET voucherStatus Example — Not Successful

The following example demonstrates the construction of an unsuccessful GET `voucherStatus` request and a response containing a `voucherStatus` object. In practice, additional HTTP headers may be included in the message.

```
Request:
GET /ssi/1.1/voucherStatus?endClientType=SSI_kiosk&endClientId=ABC_123
   &configurationId=1235813&validationId=012345678901234567 HTTP/1.1
Accept: application/json
Accept-Charset: utf-8

Response:
HTTP/1.1 200 OK
Content-Length: 139
Content-Type: application/json; charset=utf-8

{
   "endClientType": "SSI_kiosk",
   "endClientId": "ABC_123",
   "configurationId": 1235813,
   "validationId": "012345678901234567",
   "hostException": 22
}
```

NEW CHAPTER

4.6 POST redeemVoucher Resource

This resource is used to request authorization from the host to redeem a voucher.

Table 4.15 POST redeemVoucher Resource

HTTP Method	POST
Pathname	/ssi/[ver]/redeemVoucher
Request Content-Type	application/json; charset=utf-8
Request Content	redeemVoucher Object.
Response Content-Type	application/json; charset=utf-8
Response Content	authorizeVoucher Object.

4.6.1 redeemVoucher Object

The following table identifies the properties of the `redeemVoucher` Object. Additional properties MAY be included in the object.

If the request is not authorized within the time period specified in the `voucherHoldTime` configuration property, the end-client MUST return the voucher and generate a `commitVoucher` request indicating that the voucher was returned due to a timeout (`endClientException = "5"`). If an `authorizeVoucher` response is received after the voucher has been returned (or, in general, at any time a voucher is not being held in escrow), the end-client MUST simply ignore the `authorizeVoucher` response.

While waiting for the `voucherHoldTime` to expire, the end-client MUST make a best effort to retry the `redeemVoucher` request at the frequency set in the `timeToLive` configuration property until a valid `authorizeVoucher` response is received.

After generating a `redeemVoucher` request and the voucher has stacked or returned, the end-client MUST always generate a `commitVoucher` request to report the final disposition of the voucher redemption request. Even if the end-client does not receive an `authorizeVoucher` response or receives an exception in response to the `redeemVoucher` request, the end-client MUST still generate a `commitVoucher` request for the host to confirm the outcome of the redemption request.

When the `allowVoucherRedeem` configuration property is set to false, the end-client MUST NOT generate any `redeemVoucher` requests.

If the request is successful — that is, the host is authorizing redemption of the voucher in the response — the `hostException` property of the response MUST be set to 0 (zero). Otherwise, the `hostException` property MUST be set to an appropriate non-zero value; all other optional properties of the response MUST be omitted; the `configurationId`, `transactionId`, and `validationId` properties MUST be set to the corresponding values received in the request.

- If the end-client is unknown or invalid, the host MUST set the `hostException` property to `97 Unknown or Invalid End-Client`.
- If the voucher configuration is not correct, the host MUST set the `hostException` property to `21 Incorrect Voucher Configuration`.

NEW CHAPTER

- If the host determines that redemption of the voucher is already in process at another end-client, the host MUST set the hostException property to 1 Redemption in Process at Another End-Client.
- If the host determines that the voucher has already been redeemed, the host MUST set the hostException property to 2 Voucher Already Redeemed.
- If the host determines that the voucher has expired, the host MUST set the hostException property to 3 Voucher Expired.
- If the validationId is unknown or invalid, the host MUST set the hostException property to 4 Voucher Not Found.
- If the host determines that the voucher cannot be redeemed at the end-client, the host MUST set the hostException property to 5 Voucher Cannot Be Redeemed at This End-Client.
- If the host determines that the specified player is incorrect for the voucher, the host MUST set the hostException property to 6 Incorrect Player for Voucher.

A redeemVoucher request is considered logically equivalent to a previous redeemVoucher request if the host detects that the transactionId associated with the redemption request was reported in a previous redeemVoucher or commitVoucher request for the same end-client. In such cases, the host MUST generate a logically equivalent authorizeVoucher response.

Table 4.16 redeemVoucher Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
transactionId	type: t_transactionId use: required	Transaction identifier. Example, "14591424".
idReaderType	type: t_idReaderTypes use: optional default: SSI_none	ID reader type. Example, "SSI_magCard".
idNumber	type: t_idNumber use: optional default: <empty>	ID number. Example, "09900101977".
playerId	type: t_playerId use: optional default: <empty>	Player account identifier. Example, "00101977".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".
employeeAuthArray	type: array use: optional minItems: 0 maxItems: ∞	An array of employeeAuth objects. See Table 4.17,employeeAuth Object Properties for details.

Table 4.17 employeeAuth Object Properties

Property	Restrictions	Description
authCode	type: t_authCodes use: optional default: <empty>	Authorization code. Example, "SSI_authLine1".
jobCode	type: t_jobCode use: required	Job code.
employeeId	type: t_employeeId use: required	Employee identifier.

4.6.2 authorizeVoucher Object

The following table identifies the properties of the authorizeVoucher Object. Additional properties MAY be included in the object.

To authorize the redemption of a voucher, the host MUST set the voucherAmt to a non-zero value and MUST set the hostException property to 0 (zero). And, the creditType, voucherSource, largeWin, shortPay, voucherSequence, expireCredits, and expireDateTime properties MUST be set to the semantically correct values for the voucher being redeemed.

To deny redemption of a voucher, the host MUST set the voucherAmt to 0 (zero) and MUST set the hostException property to a non-zero value indicating the reason for denial. In such cases, the creditType, voucherSource, largeWin, shortPay, voucherSequence, expireCredits, and expireDateTime properties MUST be omitted or set to their default values.

The host can set the voucherSource property to SSI_system to indicate that the voucher was issued by the system (typically, for promotional purposes) or SSI_endClient to indicate the voucher was issued by an end-client, such as an EGM or kiosk. This feature can be used in jurisdictions where the expense for end-client-issued vouchers is deducted at the time of redemption. In such situations, system-issued vouchers are not deductible and, therefore, are accounted for separately from end-client-issued vouchers. In jurisdictions where this is not an issue, all vouchers can be redeemed and accounted for as end-client-issued vouchers.

The host may use the hostAction property to force an end-client to stack a voucher that is not valid or to force an end-client to return the voucher following a valid redemption. If the host authorizes redemption and the end-client is unable to redeem the voucher for any reason, the end-client MUST return the voucher regardless of the hostAction value. The host should use this attribute with extreme caution. The hostAction property may be set to one of three values:

- SSI_endClientAction tells the end-client to perform its normal action of stacking or returning a voucher; for example, stacking a redeemed voucher and returning all others.
- SSI_stack tells the end-client to stack a voucher following successful completion of the authorizeVoucher response.
  - If the host does not authorize redemption (i.e. hostException is set to a non-zero value), the end-client MUST still stack the voucher, if possible.
  - If the host authorizes redemption of the voucher and the end-client is unable to stack the voucher for any reason, the end-client MUST NOT redeem the voucher.

  - If the host authorizes redemption of the voucher and the end-client is unable to redeem the voucher for any reason, the end-client MUST NOT stack the voucher.
- SSI_return tells the end-client to return the voucher regardless of whether it was successfully redeemed or not. If hostAction is set to SSI_return, the end-client MUST NOT stack the voucher under any circumstances.

When a voucher redemption is authorized, the host MUST record that a redemption request is pending for the voucher. Until a commitVoucher request is received indicating that the voucher was not redeemed or the status of the voucher is manually reset, additional redemptions MUST NOT be permitted by the host for that voucher.

Following the generation of a redeemVoucher request, the end-client MUST always generate a commitVoucher request to report the final results of the voucher redemption request, even if no funds were transferred. If funds are transferred, the end-client should wait until the transfer is complete and the voucher has been stacked or returned before generating the commitVoucher request.

Table 4.18 authorizeVoucher Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
transactionId	type: t_transactionId use: required	Transaction identifier. Example, "14591424".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".
voucherAmt	type: t_millicents use: optional default: 0	Voucher amount. Example, "12345000".
creditType	type: t_creditTypes use: optional default: SSI_cashable	Credit Type. Example, "SSI_cashable".
voucherSource	type: t_voucherSources use: optional default: SSI_endClient	Voucher source; set to SSI_endClient. Example, "SSI_endClient".
largeWin	type: boolean use: optional default: false	Indicates whether the voucher was issued because the amount won exceeded the end-client's large win limit. Example, "false".

Table 4.18 authorizeVoucher Object Properties

Property	Restrictions	Description
shortPay	type: boolean use: optional default: false	Indicates whether the voucher was issued because the end-client could not fully redeem another voucher. Example, "false".
voucherSequence	type: numeric mode: integer use: optional default: 0 minimum: 0	The issuing end-client's internal voucher sequence number; printed on the voucher. Example, "123".
expireCredits	type: boolean use: optional default: false	Indicates whether non-cashable credits have an associated expiration date/time; ignored when creditType is not set to SSI_nonCashable. Example, "false".
expireDateTime	type: string use: optional default: <empty> format: date-time	Expiration date/time associated with non-cashable credits; ignored when creditType is not set to SSI_nonCashable or expireCredits is set to false. Example, "2001-01-01T00:00:00-00:00".
hostAction	type: t_voucherHostActions use: optional default: SSI_endClientAction	Indicates whether the host prefers the voucher to be stacked, returned, or that the client determines the appropriate action. Example, "SSI_endClientAction".
hostException	type: t_voucherHostExceptions use: optional default: 0	Host exception code. Example, "0".

4.6.3 POST redeemVoucherExample — Successful

The following example demonstrates the construction of a successful POST redeemVoucher request containing a redeemVoucher object and a response containing a authorizeVoucher object. In practice, additional HTTP headers may be included in the message.

Request:

```
POST /ssi/1.1/redeemVoucher HTTP/1.1
Content-Length: 381
Content-Type: application/json; charset=utf-8
Accept: application/json
Accept-Charset: utf-8

{
   "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "idReaderType": "SSI_magCard",
    "idNumber": "09900101977",
    "playerId": "00101977",
    "validationId": "012345678901234567",
    "employeeAuthArray": [
        {
            "authCode": "",
            "jobCode": "attendant",
            "employeeId": "1234"
        },
        {
            "authCode": "SSI_changeAmt",
            "jobCode": "manager",
            "employeeId": "2345"
        }
    ]
}
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 389
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "validationId": "012345678901234567",
    "voucherAmt": 12345000,
    "creditType": "SSI_cashable",
    "voucherSource": "SSI_endClient",
    "largeWin": false,
    "shortPay": false,
    "voucherSequence": 123,
    "expireCredits": false,
    "expireDateTime": "",
    "hostAction": "SSI_endClientAction",
    "hostException": 0
}
```

4.6.4 POST redeemVoucher Example — Not Successful

The following example demonstrates the construction of an unsuccessful POST `redeemVoucher` request containing a `redeemVoucher` object and a response containing a `authorizeVoucher` object. In practice, additional HTTP headers may be included in the message.

Request:

```
POST /ssi/1.1/redeemVoucher HTTP/1.1
Content-Length: 381
Content-Type: application/json; charset=utf-8
Accept: application/json
```

```
Accept-Charset: utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "idReaderType": "SSI_magCard",
    "idNumber": "09900101977",
    "playerId": "00101977",
    "validationId": "012345678901234567",
    "employeeAuthArray": [
        {
            "authCode": "",
            "jobCode": "attendant",
            "employeeId": "1234"
        },
        {
            "authCode": "SSI_changeAmt",
            "jobCode": "manager",
            "employeeId": "2345"
        }
    ]
}
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 164
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "validationId": "012345678901234567",
    "hostException": 4
}
```

NEW CHAPTER

4.7 POST commitVoucher Resource

This resource is used to report the final disposition of a voucher redemption request to the host. A `commitVoucher` request MUST be generated regardless of whether the voucher redemption was successful.

Table 4.19 POST commitVoucher Resource

HTTP Method	POST
Pathname	/ssi/[ver]/commitVoucher
Request Content-Type	application/json; charset=utf-8
Request Content	commitVoucher Object
Response Content-Type	application/json; charset=utf-8
Response Content	commitVoucherAck Object

4.7.1 commitVoucher Object

The following table identifies the properties of the `commitVoucher` Object. Additional properties MAY be included in the object.

- If unsuccessful and the voucher is not redeemed, the `transferAmt` property MUST be set to 0 (zero) and the `endClientException` property MUST be set to a non-zero value. The host MUST reset the status of the voucher so that it can be redeemed elsewhere.
- If successful and the voucher is fully redeemed, the `transferAmt` property MUST be set to the actual amount transferred and the `endClientException` property MUST be set to 0 (zero).
- If successful but the voucher is only partially redeemed, the `transferAmt` property MUST be set to the actual amount transferred and the `endClientException` property MUST be set to `90 Disbursement Error – Short Pay`.

The end-cleint MUST retry the `commitVoucher` request at the frequency specified in the `timeToLive` configuration property until a valid `commitVoucherAck` response is received.

The `endClientAction` property indicates the final disposition of the voucher — that is, whether the voucher was stacked or returned. If the voucher was stacked by the end-client, the `endClientAction` property MUST be set to `SSI_redeemed`; otherwise, the `endClientAction` property MUST be set to `SSI_returned`. The `endClientAction` property MUST be set based on the actual action performed by the end-client, not the action requested by the host in the `hostAction` property of the `authorizeVoucher` response.

A `commitVoucher` request is considered logically equivalent to a previous `commitVoucher` request if the host detects that the transactionId associated with the request was reported in a previous `commitVoucher` request for the same end-client. In such cases, the host MUST generate a logically equivalent `commitVoucherAck` response.

The host MUST make a best effort to acknowledge a `commitVoucher` request. Failure to acknowledge a `commitVoucher` request will cause the end-client to retry the `commitVoucher` request indefinitely and may lead to a loss of voucher-related functionality. Host exception `21 Incorrect Voucher Configuration` MUST NOT be reported in response a `commitVoucher` request. If the request is not successful, the `configurationId`, transactionId, and validationId properties MUST be set to the corresponding values received in the request.

Table 4.20 commitVoucher Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".
transactionId	type: t_transactionId use: required	Transaction identifier. Example, "14591424".
validationId	type: t_validationId use: required	Validation identifier. Example, "012345678901234567".
voucherAmt	type: t_millicents use: optional default: 0	Voucher amount. Example, "12345000".
creditType	type: t_creditTypes use: optional default: SSI_cashable	Credit Type. Example, "SSI_cashable".
voucherSource	type: t_voucherSources use: optional default: SSI_endClient	Voucher source. Example, "SSI_endClient".
largeWin	type: boolean use: optional default: false	Indicates whether the voucher was issued because the amount won exceeded the end-client's large win limit. Example, "false".
shortPay	type: boolean use: optional default: false	Indicates whether the voucher was issued because the end-client could not fully redeem another voucher. Example, "false".
voucherSequence	type: numeric mode: integer use: optional default: 0 minimum: 0	The issuing end-client's internal voucher sequence number; printed on the voucher. Example, "123".

Table 4.20 commitVoucher Object Properties

Property	Restrictions	Description
expireCredits	type: boolean use: optional default: false	Indicates whether non-cashable credits have an associated expiration date/time; ignored when creditType is not set to SSI_nonCashable. Example, "false".
expireDateTime	type: string use: optional default: <empty> format: date-time	Expiration date/time associated with non-cashable credits; ignored when creditType is not set to SSI_nonCashable or expireCredits is set to false. Example, "2001-01-01T00:00:00-00:00".
transferAmt	type: t_millicents use: optional default: 0	Actual transfer amount. Example, "12345000".
transferDateTime	type: string use: optional default: <empty> format: date-time	Date/time that the transaction took place. Example, "2016-03-31T17:11:28-5:00".
endClientAction	type: t_voucherClientActions use: optional default: SSI_redeemed	Action taken by the end-client; set to SSI_redeemed or SSI_returned. Example, "SSI_redeemed".
endClientException	type: t_voucherClientExceptions use: optional default: 0	End-client exception code. Example, "0".

4.7.2 commitVoucherAck Object

The following table identifies the properties of the commitVoucherAck Object. Additional properties MAY be included in the object.

Table 4.21 commitVoucherAck Object Properties

Property	Restrictions	Description
endClientType	type: t_clientTypes use: required	End-client type. Example, "SSI_kiosk".
endClientId	type: t_clientId use: required	End-client identifier. Example, "ABC_123".
configurationId	type: t_configurationId use: required	Configuration identifier. Example, "1235813".

Table 4.21 commitVoucherAck Object Properties

Property	Restrictions	Description

`transactionId`	`type: t_transactionId`
	`use: required`
		Transaction identifier.
		Example, "14591424".

`validationId`	`type: t_validationId`
	`use: required`
		Validation identifier.
		Example, "012345678901234567".

`hostException`	`type: t_voucherHostExceptions`
	`use: optional`
	`default: 0`
		Host exception code.
		Example, "0".

4.7.3 POST commitVoucher Example — Successful

The following example demonstrates the construction of a successful POST `commitVoucher` request containing a `commitVoucher` object and a response containing a `commitVoucherAck` object. In practice, additional HTTP headers may be included in the message.

Request:

```
POST /ssi/1.1/commitVoucher HTTP/1.1
Content-Length: 467
Content-Type: application/json; charset=utf-8
Accept: application/json
Accept-Charset: utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "validationId": "012345678901234567",
    "voucherAmt": 12345000,
    "creditType": "SSI_cashable",
    "voucherSource": "SSI_endClient",
    "largeWin": false,
    "shortPay": false,
    "voucherSequence": 123,
    "expireCredits": false,
    "expireDateTime": "",
    "transferAmt": 12345000,
    "transferDateTime": "2016-03-31T17:11:28-05:00",
    "endClientAction": "SSI_redeemed",
    "endClientException": 0
}
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 109
Content-Type: application/json; charset=utf-8

{
    "endClientType": "SSI_kiosk",
    "endClientId": "ABC_123",
    "configurationId": 1235813,
    "transactionId": 14591424,
    "validationId": "012345678901234567"
}
```

NEW CHAPTER