ORIGINAL SPIRIT

Game Catalog

v 0.9.987

Version	Description	Date
0.1.0	Draft version	03.04.2015
0.9.0	Added missing screenshots, catalog structures and game callbacks	15.05.2015
0.9.1	Added CR currency	21.01.2016
0.9.2	Added seamless mode	10.03.2016
0.9.3	Changed callback signatures	22.03.2016
0.9.4	Dropped email requirement	06.05.2016
0.9.5	Added unique id to catalog structure	29.06.2016
0.9.6	Added isRefund parameter to debit callback calls	07.07.2016
0.9.7	Added refresh callback	08.07.2016
0.9.8	Added lobby url parameter	11.07.2016
0.9.9	Added lobby documentation and new currencies	15.07.2016
0.9.91	Added clientType parameter	15.09.2016
0.9.92	Added new currencies	28.09.2016
0.9.93	Added TRY currency	15.01.2017
0.9.94	Added PLN, RUB, SEK, ZAR currencies	10.04.2018
0.9.95	Added KZT, BYN, AMD, GEL, MDL, UAH currencies	13.01.2019
0.9.96	Added LiveLobby section	16.01.2019
0.9.97	Added isRefund parameter for debit calls	22.03.2019
0.9.98	Added missing parameters for callbacks	12.06.2019
0.9.981	Fixed live lobby URL and lobbyUrl parameter	01.07.2019
0.9.982	Added remark about optional query parameters	04.07.2019
0.9.983	Added sessionId and lost bets to callback	10.11.2019
0.9.984	Added KGS currency	06.03.2020
0.9.985	Replaced frontend URLs	19.03.2020
0.9.986	Update server URLs	23.10.2020
0.9.987	Added callback API v2	28.01.2021

Table of contents

Introduction

Original Spirit game catalog is a service for integration of Original Spirit games into third party web-sites and applications. Games are HTML5-based, so they can be integrated anywhere, where HTML5 is supported – mobiles, desktops, Smart TVs, kiosks and far more. Games can run in two modes – fun mode, with no real funds involved and real mode, where Original Spirit system accepts user balance data from third party service and returns the resulted user balance after game session is complete.

This document will guide you through both software interfaces implementation, system configuration and exploitation.

General structure

Service consists of three parts:

- Game catalogs, JSON / XML documents with game list.
- Web-services for session integration
- Web-panel for control over the catalogs and user activity

System requirements

- Integrators must have HTTPS protocol enabled on their server, because Original Spirit requires HTTPS for callback functionality.
- Integrators must have static IPv4 address set for their gateway, from which they will send requests to Original Spirit systems
- Integrators must have cookies allowed for Original Spirit games in their applications/web-sites
- Integrators must have unique numeric (Int64 or less) identifier for users in their system, this identifiers will be used in Original Spirit system

Game catalog

Game catalog is a JSON document of the following structure (all data types are strings, if not specified otherwise):

Document root

- **categories** – array, root for games categories
  - …
  - **id** - category identifier
  - **title** – category title
  - **games** – array, games for this category
    - …
    - **categoryId** - parent category identifier

- **id** – unique game identifier, cannot be changed, used for seamless callbacks
- **gameId** – game class identifier
- **instanceId** – game instance identifier, can be changed during instance lifetime
- **title** – game title
- **modes** – array, available game modes
  - …
  - **Id** – mode identifier
  - **mode** - mode type (can be **REAL** (real funds involved) or **FUN** (fun funds involved))
  - **title –** mode title
  - …
- **thumbnailUrl** - url to small thumbnail (up to 320x240 pixels)
- **featuredUrl** – url to large game-related image (up to 1920x1080 pixels)
- **layout** – structure with game layout information
  - …
  - [**layoutType] –** can be "**default**"/ "**mobile**"/"**tablet**"
    - **direction** – game orientation on screen, can be **Portrait**, **Landscape** or **PortraitLandscape** for universal design.
    - **aspectRatio** – optional field with game aspect information in case game has a fixed layout, in **width**x**height** format. Sample: 4x3, 16x9.
  - …
- **loaderUrl** – base url for game loader, **including protocol prefix**. Integrators may be required to append additional parameters to it.
- …

…

Demo document for testing purposes can be found at
https://gateway.ssl256bit.com/catalogs/100014_9745650.json

Integrators own game catalogs can be added/changed/removed from web-panel.

Session integration mechanism consists of two web-services – one belongs to HollywoodTV and is located on HollywoodTV servers at https://gateway.securesocket.net/catalog_service, second one **is optional**, it belongs to integrator and is hosted on integrators server. First one provides integrators with session identifiers and methods to pass or retrieve user data. Second one is used for callback/seamless integration purposes. Both services require SSL and usage of access tokens provided by HollywoodTV system. HollywoodTV service also restricts access via IPv4 address. Usage of IPv6 is not allowed. HollywoodTV system will accept GET/POST requests with JSON documents, respond with JSON documents, send GET/POST requests to integrators callback service and expect valid HTTP responses with JSON documents as bodies from it. Content encoding of HTTP requests/responses should be utf-8, content-type – application/json.

Like said before, HollywoodTV requires token authentication, for integrator-to-HollywoodTV service requests it should be access token, which can be created in web-panel. Access token is transferred via **X-CASINO-TOKEN** HTTP header; callback token which is used for HollywoodTV-to-integrator callback is used for signing the requests.

HTTP status codes are expected to be used as response statuses. Following codes are expected:

200 – request succeeded.

400 – request failed because data provided in it is incorrect , response body is human-readable message if not empty.

403 – access denied, response body is human-readable message if not empty.

404 – requested object not found, response body is human-readable message if not empty.

409 – data conflict, response body is human-readable message if not empty.

500 – generic system error, response body is human-readable message, if not empty.

503 – HollywoodTV system is temporary not available. Call can be re-attempted later.

SERVICES ARE EXPECTED TO LOG THEIR BEHAVIOR, INCLUDING REQUESTS, RESPONSES AND MESSAGES FROM FAILED REQUESTS AND KEEP THESE LOGS AT LEAST FOR 30 DAYS.

Following data types are used in services:

- Boolean, boolean value – true/false.
- Int64, integer value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- Decimal, floating point for money representation, from -79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335.
- String, utf-8 string value.
- DateTime, string representation of date, in yyyy-MM-dd HH:mm:ss format. Always carries GMT date and time.
- Currency, 3 letter currency code, as in ISO 4217 **or** 2 letter code CR (which means Credits and represents internal casino currency). **Following currency codes are supported**: EUR, USD, CNY, ARS, VEF, MYR, THB, TRY, JPY, KRW, AUD, CAD, GBP, NZD, BRL, CHF, CZK, MXN, NOK, PLN, RUB,

SEK, ZAR, KZT, BYN, AMD, GEL, MDL, COP, UAH, GHS, TND, PHP, BDT, VND, PYG, INR, NGN, IRR and CR.

Following data structures, fields and enumerations are used in services:

UserData – holds user information

- **userId**, Int64, required – this field contains id of user in integrators system
- **username**, string, required – unique string user identifier in integrators system. This field will be used for reports purposes
- **externalSessionToken**, string, optional – session identifier from remote system, can be passed back with callback calls, 2048 chars in length max
- **nick**, string, required – this field contains common name/nick of user in integrators system. This field will be displayed in games as player identifier
- **currency**, currency, required.
- **reference**, string, required – this field contains domain for source web-site, where user came from

SessionData – holds session data for user

- **userId**, Int64, required – this field contains id of user in integrators system
- **sessionToken** , string, required – this field contains session token, which should be used by integrator system when forming game loader url
- **expirationDate**, DateTime, required – this field contains session expiration date, GMT.

BalanceData – holds user balance information

- **userId**, Int64, required – this field contains id of user in integrators system
- **masterBalance**, decimal, required – this field contains master balance
- **currency**, currency, required

TransferData – holds transfer information

- **userId**, Int64, required – this field contains id of user in integrators system
- **amount**, decimal, required – amount to transfer
- **currency**, currency, required
- **refId**, Int64, required – this field contains reference identifier for transaction. It's required to exclude the transfer duplication **– so for each unique operation it should be unique amongst the sender system.**

WithdrawData – holds withdraw information

- **userId**, Int64, required – this field contains id of user in integrators system
- **withdrawnAmount**, decimal, required – amount which was withdrawn from user balance
- **currency**, currency, required
- **balance**, BalanceData, required – this field contains resulted user balance

WinData – holds win information

- **userId**, Int64, required – this field contains id of user in integrators system
- **gameId**, string, required – this field contains game identifier

- **date**, DateTime, required – this field contains change date, GMT
- **amount**, decimal, required – this field contains win amount
- **currency**, currency, required
- **balance**, BalanceData, required – this field contains resulted user balance
- **description**, string, required – human-readable description for the win

TaxData – holds taxation data

- **taxId**, Int64, taxation id from database
- **taxTitle**, string, taxation title
- **taxAmount**, decimal, taxation amount
- **grossAmount**, decimal, gross amount of operation tax is applied to
- **netAmount**, decimal, net amount of operation

CallbackResponseData – holds user balance information, returned from callback requests

- **code**, int, optional – this field contains response status code as described below
- **balance**, decimal, required – this field contains user balance after callback operation
- **message**, string, optional – this field contains optional message to display for player

ClientType – is a string enumeration, which holds client machine type. Can be one from the following list: desktop, mobile, TV, tablet, other.

First service, the one in HollywoodTV system, exposes the following methods:

Method name and signature	Service relative path	HTTP verb	Description
`SessionData setUserData(UserData userData);`	/set_user_data	POST	This method will accept **userData**, update or create session and return its information to caller.
`SessionData getUserSession(long userId);`	/get_user_session?userId= **userId**	GET	This method will return active session information for provided **userId**, if such exists. If not, new session won't be created and server will respond with HTTP status code 404.
`BalanceData getUserBalance(long userId, [string currency]);`	/get_user_balance?userId= **userId**[&currency=**currency**]	GET	This method will return user balance information for provided **userId**, if there is an existing user account with matching optional **currency** parameter (or last used/currently used currency, if **currency** parameter is not

			provided). If no matching account exists, server will respond with HTTP status code 404.
`BalanceData deposit (TransferData transferData);`	/deposit	POST	This method will perform deposit to HollywoodTV account, associated with **userId** and return the resulted user balance. If operation with such **refId** already exists in our system, no action will be performed.
`WithdrawData withdraw(TransferData transferData);`	/withdraw	POST	This method will perform withdraw from HollywoodTV account, associated with userId and return the resulted withdraw result. If operation with such **refId** already exists in our system, no action will be performed, and withdraw result will contain data for previous operation with such **refId**.
`WithdrawData withdrawAll(TransferData transferData);`	/withdraw_all	POST	This method will reset user balance, if any – by withdrawing from it to threshold value – and returning the result value. **amount** field is ignored. If operation with such **refId** already exists in our system, no action will be performed, and withdraw result will contain data for previous operation with such **refId**.

Second service, the one in integrator system, exposes the methods to accept cashier: balance, debit and credit. They will be called in case seamless mode is configured in access point settings. Requests are simple POST requests, all parameters are passed as JSON document, action is specified as **action** parameter in document. Response is expected to be a valid JSON document with successful HTTP code, or HTTP code with relation to the failure reason – 400 if service can't understand the data or check hash, 404 if user not found, 403 if player can't be debited, 500 for system errors. It is also possible to use HTTP 200 code for all response and provide result code in the JSON response (via code property). Each query

is signed with SHA-256 hash (HTTP header **X-CASINO-TOKEN**, hexadecimal string) of JSON document, calculated as following: **SHA256(callback_token<newline>request body).**

CALL BACK QUERIES MAY CONTAIN ADDITIONAL PARAMETERS, WHICH SHOULD BE IGNORED.

1. Balance callback call, should retrieve actual user balance:

{

"action": "balance",

"refId": [long, identifier of access point where call originated from],

"accountId": [long, user identifier provided on session creation],

"gameId": [long, game instance identifier],

"clientType": [string, of ClientType enumeration],

"currency": [string, currency identifier],

"sessionToken": [string, optional session identifier inside the system, returned on set_user_data call],

"username": [string, optional username provided on session creation],

"externalSessionToken": [string, optional integrator session identifier provided on session creation]

}

2. Debit callback call, called when user made bets - and for rollback on winnings:

{

"action": "debit",

"refId": [long, identifier of access point where call originated from],

"accountId": [long, user identifier provided on session creation],

"gameId": [long, game instance identifier],

"currency": [string, currency identifier],

"amount": [decimal, gross amount of the bet, can be 0 for refund calls (for refund calls amount should be taken from source operation)],

"roundId" [long, identifier of the draw],

"transactionId", [long, unique identifier for the operation. **Integrators must check existing identifiers in their database to avoid operation duplication. For repeating calls with same transactionId no transaction should be made, actual balance for the user should be returned.],**

"clientType": [string, of ClientType enumeration],

"sessionToken": [string, optional session identifier inside the system, returned on set_user_data call],

"username": [string, optional username provided on session creation],

"externalSessionToken": [string, optional integrator session identifier provided on session creation],

"taxData": [TaxData, optional, can be null for refund calls],

"extData": [game-specific object with bet details, can be null for refund calls],

"isRefund": [boolean, appears only when the call is a rollback/refund call. **Integrators must check that credit transaction with such identifier exists in their database and only then perform rollback. If such transaction doesn't exist, they shouldn't change user's account and simply return HTTP 404 response.**]

}

3. Credit callback call, called for winnings, lost bets:
{
"action": "credit",
"refId": [long, identifier of access point where call originated from],
"accountId": [long, user identifier provided on session creation],
"gameId": [long, game instance identifier],
"currency": [string, currency identifier],
"amount": [decimal, gross amount of the bet, will be 0 for lost bet calls (configured in access point settings)],
"roundId" [long, identifier of the draw],
"transactionId", [long, unique identifier for the operation**. Integrators must check existing identifiers in their database to avoid operation duplication. For repeating calls with same transactionId no transaction should be made, actual balance for the user should be returned.**],
"sourceTransactionId", [long, optional identifier for the debit call connected with this credit call.],
"clientType": [string, of ClientType enumeration],
"sessionToken": [string, optional session identifier inside the system, returned on set_user_data call],
"username": [string, optional username provided on session creation],
"externalSessionToken": [string, optional integrator session identifier provided on session creation],
"taxData": [TaxData, optional, can be null for refund calls],
"extData": [game-specific object with bet details, can be null for refund calls],
}

4. Refund debit call, called for rollback of bets:
{
"action": "debitRefund",
"refId": [long, identifier of access point where call originated from],
"accountId": [long, user identifier provided on session creation],

"gameId": [long, game instance identifier],
"currency": [string, currency identifier],
"roundId" [long, identifier of the draw],
"transactionId", [long, unique identifier for the operation to refund**. Integrators must check that debit transaction with such identifier exists in their database and only then perform rollback. If such transaction doesn't exist, they should just return code 404**.],
"clientType": [string, of ClientType enumeration],
"sessionToken": [string, optional session identifier inside the system, returned on set_user_data call],
"username": [string, optional username provided on session creation],
"externalSessionToken": [string, optional integrator session identifier provided on session creation],
}

5. Refund credit call, called for rollback of wins:
{
"action": "creditRefund",
"refId": [long, identifier of access point where call originated from],
"accountId": [long, user identifier provided on session creation],
"gameId": [long, game instance identifier],
"currency": [string, currency identifier],
"roundId" [long, identifier of the draw],
"transactionId", [long, unique identifier for the operation to refund. **Integrators must check that credit transaction with such identifier exists in their database and only then perform rollback. If such transaction doesn't exist, they should just return code 404. If integrators don't support refunding wins, they should return code 403**.],
"clientType": [string, of ClientType enumeration],
"sessionToken": [string, optional session identifier inside the system, returned on set_user_data call],
"username": [string, optional username provided on session creation],
"externalSessionToken": [string, optional integrator session identifier provided on session creation],
}

Localization

By default game catalog document and games themselves will have language set in related casino entity. But integrator can request other languages, by appending parameter **langId** to catalog and game loader URLs. **langId** should contain two-letter language code, as is ISO 639-1. If language not supported, system will fall back to default casino language.

Game callbacks

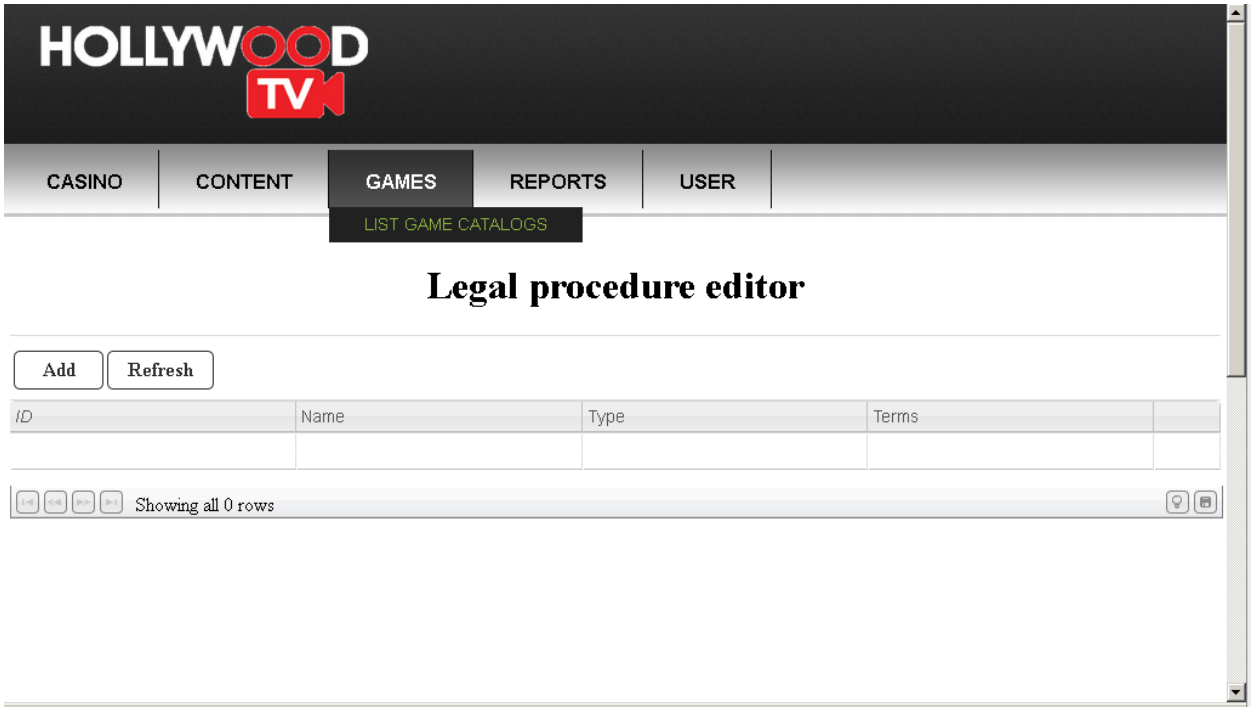
Integrators can use window callbacks to retrieve in-game state and error. Currently only four events are available in following manner:

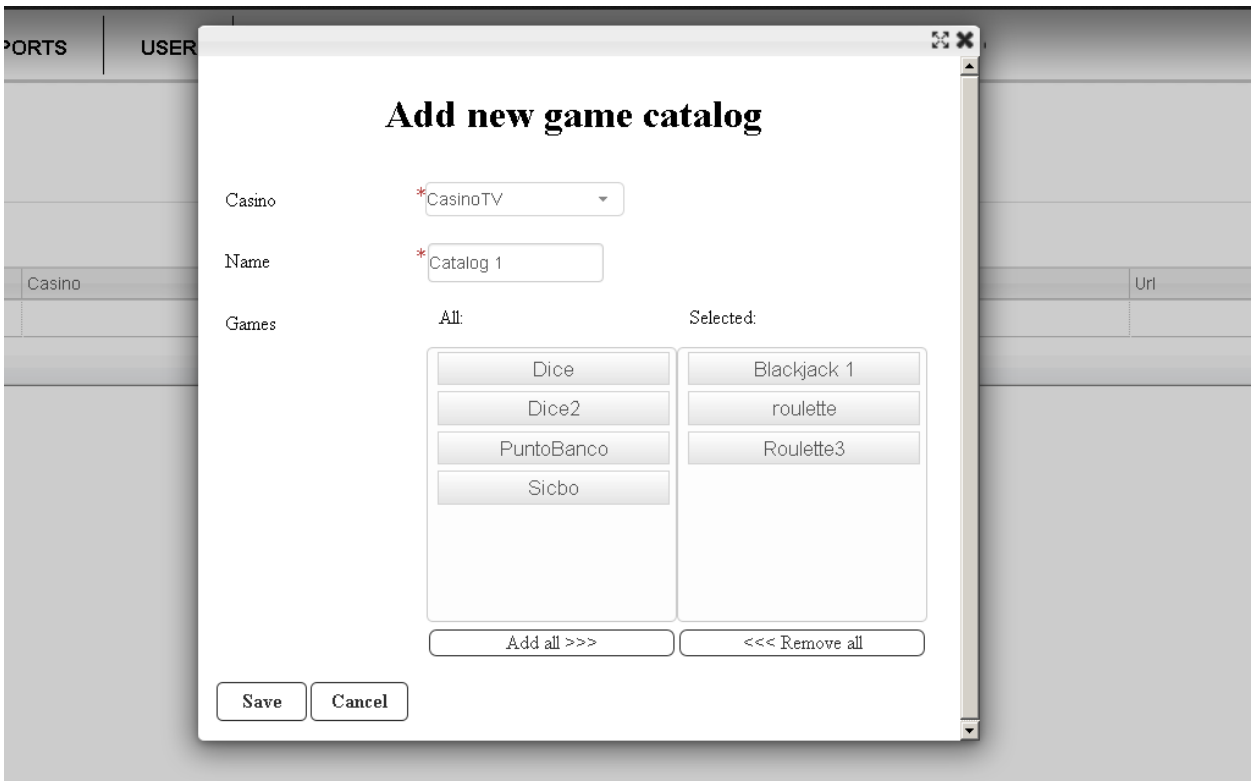
```
function listener(event) {
   if (event.data.type == "error") {
      ... read and process event.data.message …
   } else if(event.data.type == "loadSuccess"){
      ... process game loaded event…
   } else if(event.data.type == "cashier"){
      ... request for cashier form…
   } else if(event.data.type == "home"){
      ... redirect to main page (event appears, if lobbyUrl parameter wasn't provided for loader url)…
   } else if (event.data.type === "refresh") {
      ... reload the game (session token should be refreshed too)…
   }
}
if (window.addEventListener) {
   addEventListener("message", listener, false);
} else {
   attachEvent("onmessage", listener);
}
```

System setup

After integrators implemented catalog and services support in their system, they need to set up Original Spirit system in order to test, and the for production purposes.

First step is opening the Original Spirit customer panel (integrator should have username and password at this step) and opening game catalog editor. Here integrators can add new catalog with all the games they want to be in it.

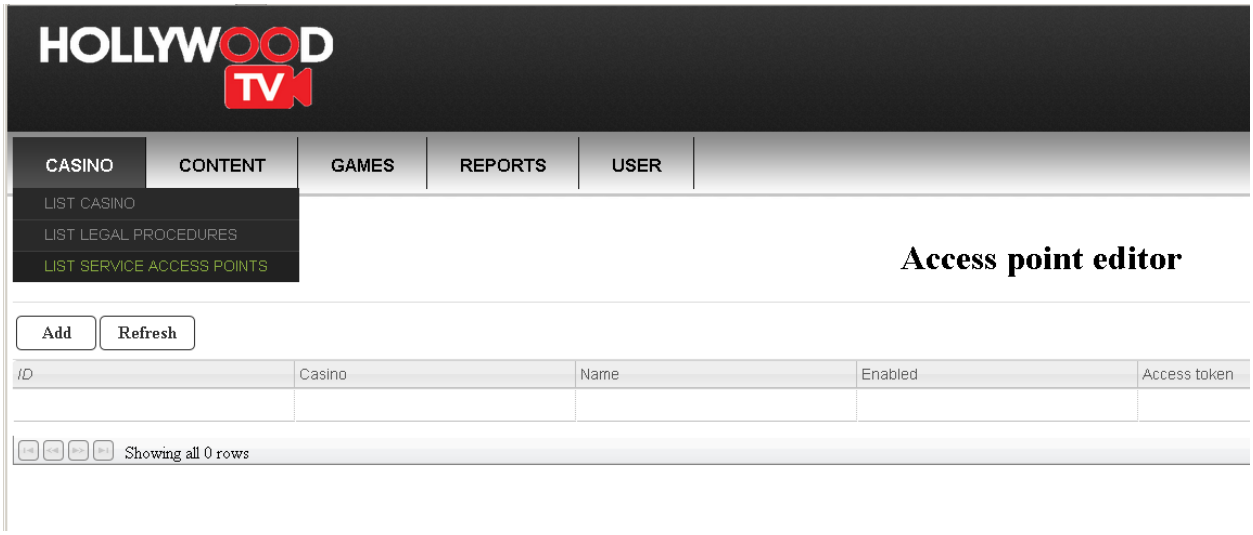


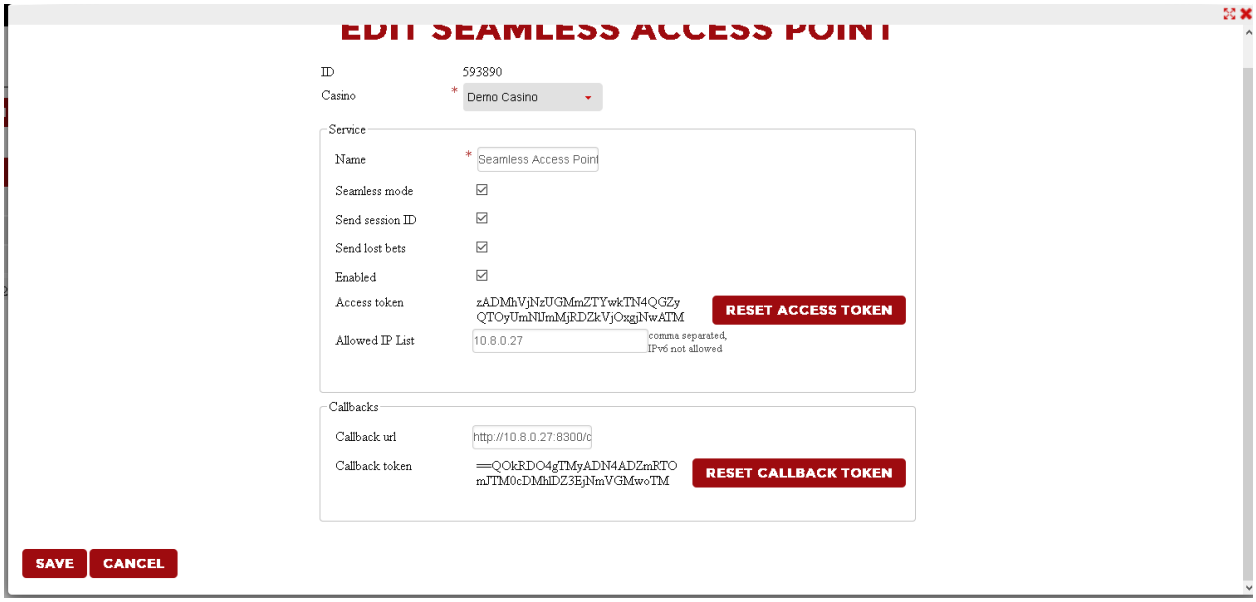


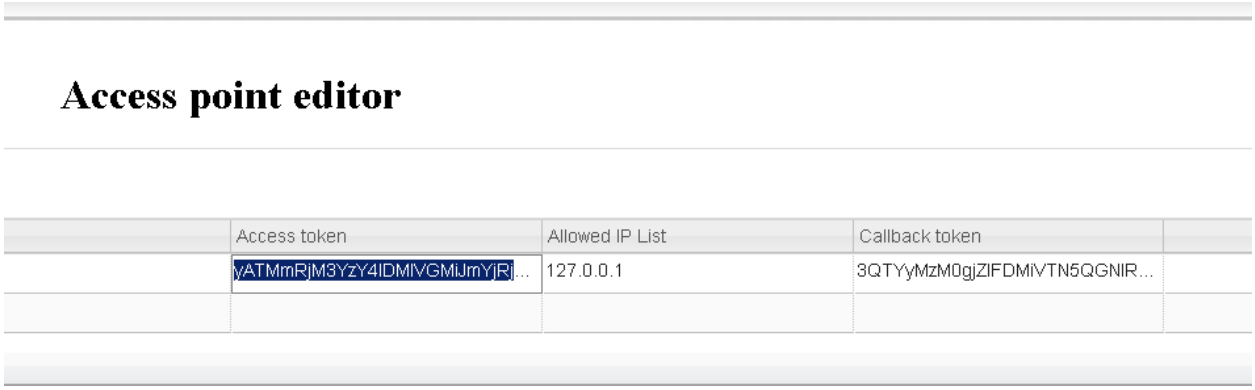
Game catalog editor

	Name	Url
	Catalog 1	http://gateway.casinotv.com/catalogs/100002_3275993.json

Second step is opening access point editor and adding the access point for the service. Here integrators can get access token for Original Spirit server and set up optional callback service.







After this integrators can start implementing, testing the integration – and afterwards get it into production.

Service implementation

- Integrators request game catalog, parse it and present as game lobby for the client
- When user selected game and mode, integrators should load the game by appending the selected mode as **mode=mode.id** parameter and using this url in target game container. Integrators should use **layout** property of game to find out the required container layout.
    - If no mode provided, Original Spirit system will choose the default mode – fun if user is anonymous, real if user is logged in
    - If user is logged in and there is no active – or there is an expired (can be requested via **getUserSession** method) Original Spirit session, integrator have to call **setUserData** once prior to game loading to request the session token. This session token should be appended as **sessionToken** parameter to the loader url
    - To transfer funds into Original Spirit system integrators should call **deposit** function **(this step is omitted, if access token is configured for seamless integration, seamless callbacks will be used instead)**
- After user completed gaming, integrator system should call one of the withdrawal methods to withdraw resulted amount from Original Spirit **(this step is omitted, if access token is configured for seamless integration, seamless callbacks will be used instead)**
- For anonymous users there is no need to append any session parameters or perform any session calls
- Integrators may append lobbyUrl parameter to the loader url. If appended it will be used instead of home event. This parameter should hold the url of casino game lobby or home page.

Lobby

Integrators can use Original Spirit ready lobby for integrations with 3d-party web-sites instead of processing the catalogs by themselves.

Lobby is located at https://g.ssl256bit.com/_Apps/lobby/ and has following parameters:

- **catalogId**, required – catalog identifier, which to use for lobby generation. Say, for full catalog of public games the resulting URL will be https://g.ssl256bit.com/_Apps/lobby/?catalogId=100092_3685299
- **lobbyUrl**, required - parameter should hold the url of casino game lobby or home page.
- **sessionToken**, optional – session token, received from Original Spirit system after session creation. If provided, games from lobby will be loaded with this token. If not provided, games in lobby will run in fun/anonymous/demo mode.

Lobby is expected to be loaded in frame and uses same window callback scheme as games do, so integrators can use the callbacks to process events like **refresh** or **home**. Event refresh is the most important and will be called on session expiration, so integrators may use it to regenerate **sessionToken** and reload the lobby afterwards. Event home will be called, when user returns from lobby to main site page.

User activity reports

Integrators can track users via Reports section in customer panel. Username, provided via **setUserData** function will be used for report generation.