
NetEnt CasinoModule™ 10.10
Integration Guide

Revision Table

Version	Description	Date
1.0	NetEnt CasinoModule™ 10.10 Integration Guide	7 April 2017

Table of Contents

CasinoModule™ Integration Guide ... 3
Integration Process ... 4
Integrate CasinoModule™ ... 6
Customer Specific Environment URLs ... 6
Customer Dedicated Test Environment URLs and Parameters ... 6
Customer Production Environment URLs and Parameters ... 7
Casino Lobby Design ... 7
Log In and Register Users ... 8
Methods ... 8
Parameters ... 9
Deposits ... 11
Deposit Money (Basic Wallet Mode) ... 11
Withdrawals ... 13
Withdraw Money (Basic Wallet Mode) ... 13
Change Coin Values for Games ... 14
Display Current Activity ... 14
Jackpots ... 15
Display Jackpot Information ... 15
Prevent Fraud and Irregularities in Jackpot Games ... 15
Publish Jackpot Terms and Conditions ... 16
Display Game History ... 16
Unfinished Game Rounds ... 17
Publish Unfinished Game Rounds and Session Timeouts Terms and Conditions ... 18
Affiliate Programs ... 18
Register Affiliate ... 19
Associate Player with Affiliate ... 20
Multi-tier Affiliate System ... 21
Tournaments ... 21
Tournament Lobby Recommendations ... 21
Tournament Details Page Recommendations ... 21
Play Game in Tournament ... 21
Tournament Methods in SOAP APIs ... 22
SOAP WS-I Compatible SOAP API ... 22
Display List of Available Tournaments ... 22

Display Tournament Details ... 23
Leaderboard ... 23
Leaderboard Exclusion Threshold ... 23
Player's Name Display ... 23
Publish the Leaderboard ... 23
Publish and Customize with HTML5, Javascript and CSS ... 23
Widget Javascript Code Example ... 26
Widget CSS Code Example ... 32
Extended Tournament Support ... 34
Bonus Programs ... 34
Wallet Options and Bonus Programs ... 34
Withdraw Money and Deposit Bonus Programs ... 34
The getUserBonusInfoV4 Method ... 35
Deposit Bonus Scenarios ... 36
Inform Players about Bonus Withdrawal Restrictions ... 36
Free Rounds Bonus Programs ... 37
Bonus Program Promotion Codes ... 37
Frequent Player Points (FPP) Program ... 38
FPP Program Functionality ... 38
Configure FPP Points Awards ... 39
Recommended FPP Program Integration Process ... 39
Manage Play and Block Limits ... 40
Currency Exchange Rates ... 41
Manage Currency Exchange Rates ... 41
Player Currency Conversion ... 42
Convert Player to new Player Currency ... 42
Integration Reference ... 43
SOAP API ... 43
WSDL Files ... 43
SOAP API Documentation ... 43
Operator and Merchant Identification ... 44
Response Caching ... 44
Response Formats ... 44
Handle Exceptions ... 44
Appendix: Migrate to WS-I Compatible SOAP API ... 45

CasinoModule™ Integration Guide

This user guide provides information about how to integrate CasinoModule with an operator's systems.

This user guide consists of the following parts:

- Overview of the integration process and detailed information about main activities in the integration process.
- Detailed description of activities to be performed during the integration process, for example setting up various casino functions such as casino lobby design, jackpots and tournaments.

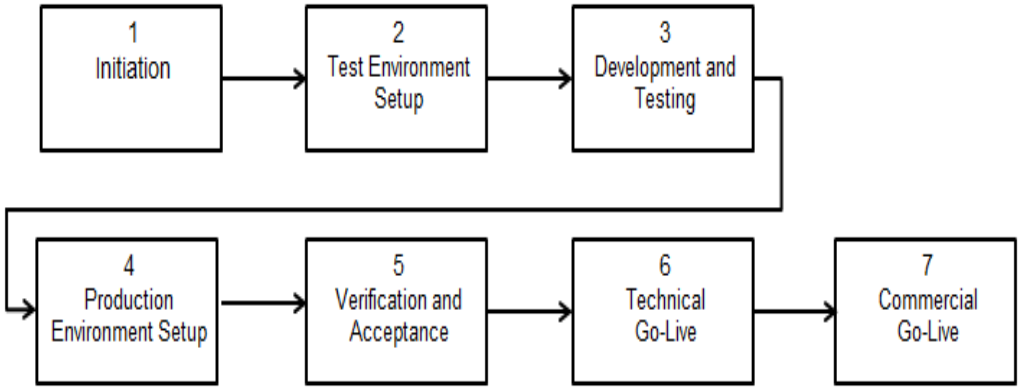
Note:

For information about integrating the games, see the game integration guides (available on the **Client Area** on the NetEnt website).

Integration Process

The integration of a new operator, or a new CasinoModule instance for an existing operator, is a process triggered by signing an agreement. The process ends when the new integration is in live operation for the operator.

The main activities in the CasinoModule integration process are shown below.



Workflow

Below is a summary of the steps involved in the integration process.

#	Actions in the process between NetEnt and the Partner	Responsible
1.	**Start-up:** ■ NetEnt assigns Account Manager. ■ NetEnt assigns Technical Integration Manager. ■ NetEnt sends integration introduction letter with contact details and how to gain access to the **Client Area** of the NetEnt website. ■ NetEnt arranges technical start-up/process overview meeting with partner.	Head of Account Management. NetEnt Customer Integrations Team Lead NetEnt Technical Integration Manager
2.	NetEnt sets up test environment and provides access to test environment for partner.	NetEnt Technical Integration Manager
3.	**System integration process:** Partner starts the system integration by implementing the API methods, etc.	Partner
4.	When the partner indicates they are ready to begin integration verification the NetEnt Technical Integration Manager orders the dedicated customer production environment.	NetEnt Technical Integration Manager

#	Actions in the process between NetEnt and the Partner	Responsible
5.	Partner tests and verifies the integration in alignment with the NetEnt Technical Integration Manager. NetEnt and partner agree on 'technical go-live' date.	Partner and NetEnt Technical Integration Manager
6.	NetEnt provides access to production environment for technical go-live.	NetEnt Technical Integration Manager
7.	Partner opens to real money players. System is commercially live!	Partner

Integrate CasinoModule™

This topic provides information on how to set up the various casino functions required when integrating CasinoModule with your system.

Important:
Your specific jurisdiction may not support all CasinoModule features described in this section.

Customer Specific Environment URLs

The following tables provide the URLs and parameters for the operator dedicated test environment and production environment.

Customer Dedicated Test Environment URLs and Parameters

Dedicated Test Environment	Description
CasinoModule WS-I Compatible SOAP API `https://<casinoid>-api-test.casinomodule.com/ws-jaxws/services/casino`	The end point for the CasinoModule SOAP WS-I Compatible SOAP API. This API provides the backend web services to a operator system.
CasinoModule WS-I Compatible SOAP API WSDL `https://<casinoid>-api-test.casinomodule.com/ws-jaxws/services/casino?wsdl`	The WSDL description of the CasinoModule SOAP WS-I Compatible SOAP API.
Game Services URL `https://<casinoid>-game-test.casinomodule.com/game/`	The location from which game rules and game history are loaded.
Game Load URL `https://<casinoid>-static-test.casinomodule.com/`	The location from which static game content for Flash-based games is loaded.
Game Play URL `https://<casinoid>-game-test.casinomodule.com/`	The location from which dynamic game content is loaded during play.
ADMIN TOOL `https://<casinoid>-admin-test.casinomodule.com/admin/`	CasinoModule ADMIN TOOL.

Note:
`CasinoId` is unique per operator and is assigned during the integration project.

Customer Production Environment URLs and Parameters

Production Environment	Description
CasinoModule WS-I Compatible SOAP API `https://<casinoid>-api.casinomodule.com/ws-jaxws/services/casino`	The end point for the CasinoModule SOAP WS-I Compatible SOAP API. This API provides the back-end web services to a operator system.
CasinoModule WS-I Compatible SOAP API WSDL `https://<casinoid>-api.casinomodule.com/ws-jaxws/services/casino?wsdl`	The WSDL description of the CasinoModule SOAP WS-I Compatible SOAP API.
Game Services URL `https://<casinoid>-game.casinomodule.com/game/`	The location from which game rules and game history are loaded.
Game Load URL `https://<casinoid>-static.casinomodule.com/`	The location from which static game content for Flash-based games is loaded.
Game Play URL `https://<casinoid>-game.casinomodule.com/`	The location from which dynamic game content is loaded during play.
ADMIN TOOL `https//<casinoid>-admin.casinomodule.com/admin/`	CasinoModule ADMIN TOOL.

Note:

`CasinoId` is unique per operator and is assigned during the integration project.

Casino Lobby Design

The purpose of the operator's casino lobby is to:

- Provide the cashier function through which players manage the funds in their accounts.
- Provide the links that launch CasinoModule games. For information about integrating the games, see the game integration guides (available on the **Client Area** on the NetEnt website).
- Provide information about CasinoModule tournaments and bonuses.

The operator's casino lobby can also be used to group the games according to their type. In NetEnt's own showcase lobby (http://www.netent.com/games), games are grouped according to their type: table games, slots, video pokers, and so on. Operators are of course free to group and list game links in any way they see fit. However, we offer the following advice:

- In Basic wallet implementations, pages that link to basic games should provide easy access to the cashier function that the player will use to transfer funds to the CasinoModule wallet.
- It is advisable to deploy a separate tournament lobby where players can view scheduled tournaments.

Note:

To limit the risk of having to make a major payout, games with large maximum wins, like The Reel Steal and High-Roller Blackjack, can be excluded from the list of games in your casino lobby pages. Your Account Manager will recommend which games are most suitable for the size of your business.

Log In and Register Users

This topic provides information about the methods and parameters used to log in and register players.

The CasinoModule SOAP API provides the following methods for logging in and registering users:

Method	Returns
`loginUser`	Logs in (and, if necessary, automatically registers) the player to play browser-based games. Returns a unique sessionId. For more information see loginUser.
`loginUserDetailed`	Logs in (and, if necessary, automatically registers) the player to play browser-based games, passing detailed demographic information such as country and gender. Returns a unique `sessionId`. Ensure that the correct channel key is used. For more information see loginUserDetailed.
`registerUser`	Registers the player, passing username and detailed demographic information such as country and gender. For more information see registerUser.

Methods

The `loginUserDetailed` and `loginUser` methods accept any unique identifier as the player's userName, such as the player's email address, player identification number, userId within the operator's database, etc.

Important:

To simplify later troubleshooting of player-related issues, we strongly recommend you pass the identifier that is most familiar to your Customer support representatives. For the same reason, we recommend that the ordinary first name and last name of the player and any other necessary identification details are passed within the `loginUserDetailed` call.

If a player is already logged in when you execute a login method, the existing `sessionId` will be returned, and no changes to the demographic information will be stored. Neither will bonus programs or other functionality, that usually runs when logging in, run.

loginUser

The `loginUser` method is used to log in a player but omit the player's demographic information. This method is preferred if demographic information is not needed or required by jurisdiction or market regulations.

Important:

The `loginUser` method must **not** be used if the player once has been registered with a jurisdiction code in the demographic information. The reason for this is that the registered jurisdiction code will be lost if the jurisdiction code is not provided with all subsequent logins for this player. To avoid that the registered jurisdiction code will be lost, use the `loginUserDetailed` method instead.

As it is unlikely that details such as country change very often, save the other methods (`loginUserDetailed` and `registerUser`) for when demographic information has actually changed or when you have a registration process that is separate from logging in, use `registerUser` to register the player.

If the player is not already registered, this method will automatically perform the registration.

loginUserDetailed

The `loginUserDetailed` is used to log in the player and at the same time pass detailed demographic information such as country, gender, jurisdiction, etc.

Important:

For most markets the parameter `JurisdictionCode` is optional, but some markets may require that the `JurisdictionCode` in the player's demographic information is passed for the player to be able to log in. For more information, see Jurisdiction Code.
The `loginUserDetailed` method **must** be used if the player once has been registered with a jurisdiction code in the demographic information. The reason for this is that the registered jurisdiction code will be lost if the jurisdiction code is not provided with all subsequent logins for this player.

For `loginUserDetailed`, you may use a `DisplayName` that is different from the player's `userName`. The `DisplayName` is used in the user interface in for example tournament leaderboards.

If the player is not already registered, this method will automatically perform the registration.

registerUser

The `registerUser` method registers the player with CasinoModule without logging the player in. It includes demographic information about the player such as country, gender, jurisdiction, etc. This method could be used if demographic information is needed or required by jurisdiction or market regulations.

Important:

For most markets the parameter `JurisdictionCode` is optional, but some markets may require that the `JurisdictionCode` in the player's demographic information is passed for the player to be able to log in. For more information, see Jurisdiction Code.

Remember that on subsequent logins for this player, you must use the same `userName` and the same user password.

Tip:

You can also use the `registerUser` method to delay the first login event until the player has launched a game, which will result in statistics based on the first login date. This is not recommended for mini games. Another reason for using this method could be, for example, when the player first registers with your website.

Parameters

User name

The parameter `userName` is case insensitive, the CasinoModule regards the user names 'FREDblogs', 'FredBlogs', and 'fredblogs' as identical. `userName` has a maximum length of 20 characters.

The following characters are supported in user names:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

0123456789

! # % ( ) = - _ @ ' * + |
```

Unsupported characters are filtered out for security or compliance reasons and using them will cause ADMIN TOOL and CasinoModule to malfunction. For example, it will not be possible to look up the player by user name, or add the user name to bonus programs, etc.

Important:
It is strongly recommended for operators to build their integrations in order to not allow any unsupported characters in user names.

Country

A two-letter ISO-3166 country code must always be used when this parameter is passed.

Jurisdiction Code

In most cases the `JurisdictionCode` parameter is optional to use, but some market regulations may require that the `JurisdictionCode` is passed with the `loginUserDetailed` or `registerUser` call. A jurisdiction code must always be used when this parameter is passed. The `JurisdictionCode` parameter has a maximum length of 4 characters. The following jurisdiction codes are supported by CasinoModule.

Jurisdiction	NetEnt Code
Alderney	ALD
Belgium	BEL
Bulgaria	BGR
Czech Republic	CZE
Denmark	DEN
Estonia	EST
Gibraltar	GIB
Isle of Man	IOM
Italy	ITA
Latvia	LAT
Malta	MLT
Netherlands	NED
Schleswig Holstein	SHH
Serbia	SRB
Spain	SPN
United Kingdom	UK

Operator Code

If you don't want to use the jurisdiction codes provided by CasinoModule it is possible to create your own combination of characters that will represent each jurisdiction. The `OperatorCode` parameter has a maximum length of 50 characters.

Server Token

The optional `ServerToken` parameter in the `loginUserDetailed` call supports inclusion of token strings that are unique per player session. According to NetEnt's session management, which is one session per channel, this means that a player playing simultaneously on mobile and desktop will have two separate tokens active. Therefore it is important to always return separate tokens for different channels.

The token is received from the operator server as part of the initial `loginUserDetailed` response, and then passed back to the operator server in the Seamless Wallet calls. However, the system supports receiving a new token with any response from the operator server, which means that the operator can switch a player's token any time.

Once a token is set for a player, the system will send this token with each `withdraw` and `deposit` Seamless Wallet call made to the operator server. The same token will be reused, until a new one is set by the operator server, or existing one is removed - in a new `loginUserDetailed` call. If the operator server sends an empty string as a token, the token is automatically removed, so that future calls do not contain any tokens.

Note:
This feature is supported only for Seamless Wallet API calls which are made in a game round context, as opposed to for example tournament prize payouts.

Server token scenarios and rules:

1. The player starts a game and is logged in through the `loginUserDetailed` method containing the `ServerToken` parameter.
2. The system stores the token for the player session.
3. In the Seamless Wallet calls (withdraw and deposit), the token is retrieved for the particular player and session.

   The system manages the server token according to these rules:

   - If token is the same as before, then no action is done. The existing token is used.
   - If new token is received, then the token is updated.
   - If token is empty or NULL, then no action is done.

Deposits

This topic contains information about transferring money in Basic wallet mode.

Deposit Money (Basic Wallet Mode)

When depositing money, funds (up to a maximum of 15,000,000 units of currency) are transferred from a player's account (wallet) on the operator's site to his corresponding account with CasinoModule.

Note:
Deposit and withdrawal operations are critical transactions in the communication between the operator system and CasinoModule. The operator is responsible for this integration, and for verifying that the interaction between the systems is conducted in accordance with this manual.

Workflow

A deposit of money *must* be made as a two-step process by the operator's system.

#	Description
1.	Initiate the transaction and communicate with CasinoModule
2.	Verify the data returned by CasinoModule and finalize the transaction.

This avoids a situation where communication problems during a transaction between the operator's system and CasinoModule causes money to be "lost" or duplicated without tracking possibilities.

Methods

The following methods are related to deposit money into the casino and should be considered before implementing the deposit functionality.

For more details see the *CasinoModule™ SOAP API* documentation on the **Client Area** of the NetEnt website.

Method	Description
`depositUserMoneyV3`	Deposits money into a player's CasinoModule account.
`depositUserMoneyBypassDepositBonusProgramsV3`	Deposits money into a player's CasinoModule account without triggering any deposit bonus programs.
`getDepositBonusAmountForPlayer`	Calculates deposit bonus amount that player will receive if the amount passed is deposited.

Follow the procedure below to deposit money.

Step by Step

1. Verify that the link to CasinoModule is alive by calling the `ping` method.
2. Initiate a withdrawal operation from the player's account with the operator.

   During this operation, deduct the money from the operator account and set the record of the withdrawal to 'initiated' or 'in process' in your database. Commit this database transaction. During this database transaction, a unique transaction ID should be generated, typically as an integer counter or identity field.
3. Call the `depositUserMoneyV3` method, passing the authentication parameters, the player's user name, the deposit amount, and the unique transaction ID generated in

the previous step. These methods add the amount to the player's CasinoModule account and returns a unique deposit ID to the operator's system.

4. Verify that the transaction ID is not displayed for any other transactions in the local transaction tables (the deposit ID returned by CasinoModule must be unique on the operator's side as well).
5. When you have verified that ID is unique, store the returned deposit ID with the withdraw transaction and set the transaction to a 'completed' state.

Note:

These methods may throw exceptions. You must perform tests to ensure that all exceptions are handled correctly. Exceptions are thrown if:
- The player cannot be found.
- The `transactionId` has already been used in a deposit or withdrawal.
- The `depositionId` returned by CasinoModule is not unique.

Important:

If communication problems occur during the deposit call (which is unlikely as a `ping` test has been made), the call will either throw an exception or time out. The operator is responsible for handling failures of deposit and withdrawal and should therefore keep track of all problems that occur for these transactions. Whether the communication broke before reaching CasinoModule or during the reply, it can be detected by tracing the transaction by the operator transaction ID using CasinoModule ADMIN TOOL. Appropriate action can be taken (usually cancelling the transaction on the operator's side, or setting it to 'completed').

Withdrawals

This topic contains information about transferring money in Basic wallet mode.

Withdraw Money (Basic Wallet Mode)

When withdrawing money, funds are transferred from a player's CasinoModule account to the players account (wallet) on the operator's site. To do this, you use `withdrawUserMoneyV3`.

The withdraw operation is basically the reverse of the deposit operation, with some minor differences, and the same safety precautions apply. It is imperative that a withdrawal, like deposit, is made as a two stage process.

Note:

Deposit and withdrawal operations are critical transactions in the communication between the operator system and CasinoModule. The operator is responsible for this integration, and for verifying that the interaction between the systems is conducted in accordance with this guide. If bonus programs are in operation, the player may not be able to withdraw the entire account balance if some of that balance is bonus money on which there are wagering requirements.

NetEnt recommends that each operator adheres to the following procedure when a player requests a certain amount to be transferred from his CasinoModule account to his operator account.

Follow the procedure below to withdraw money.

Step by Step

1. Verify that the link to CasinoModule is up by calling the `ping` method.
2. Initiate a deposit operation on the player's account with the operator.
   No money is added to the account at this point. The withdrawal is set to an 'initiated' or 'in process' state in the database. Generate a unique transaction ID.
3. Call the `withdrawUserMoneyV3` method, passing authentication parameters, the player's user name, the amount to withdraw, and the transaction ID generated in the previous step. If the player has sufficient funds in his CasinoModule account, the requested amount is deducted from the account, and a unique CasinoModule withdraw ID is returned.
4. The operator's system must verify that the returned withdraw ID is a positive integer and that it is unique (which means that it has not been used in any previous withdrawals).
   No money may be added to the player's operator account until this has been confirmed. Only when the withdraw ID has been verified should the operator system add the requested amount to the player's operator site account, set the transaction to a 'completed' state, and store the CasinoModule withdraw ID. Keep this operation in a single database transaction if possible.

Note:

The `withdrawUserMoneyV3` method may also throw exceptions. It is imperative that the operator system handles these exceptions correctly. Exceptions are generated if:
- The player cannot be found or if an incorrect password was specified.
- The transaction ID has already been used in a withdrawal or deposit.
- The player does not have the requested amount in his CasinoModule account.

Change Coin Values for Games

CasinoModule games support multiple currencies.

For each supported currency, the coin values have been carefully chosen to match the payout and hit probabilities for each game and do not normally need to be changed. For information about the currency values (for example, chip values or coin values) per game, contact your Technical Integration Manager.

Contact our Customer Support team who can advise you on the values that can be configured and who can change the values for you: support@netent.com.

Display Current Activity

The following SOAP API method provides information about current activities in a form, suitable for display in scrolling banners, tickers, and announcements:

Method	Returns

`getCurrentOverview`	Information about current activity, suitable for display in a ticker, including information like:
	■ current number of players
	■ the most popular game
	■ tournament information
	■ winners information (return *n* last wins over *n* amount where *n* can be configured by NetEnt Customer Support):
	  ■ `winTime`
	  ■ `winUserName`
	  ■ `winCountry`
	  ■ `winAmount`
	  ■ `winGameId`
	■ Jackpot information - get all jackpot amounts in one call.

For methods related to jackpot information specifically, see Display Jackpot Information.

Jackpots

Casino players are enticed to bet by seeing other players win. Displaying current jackpot levels, information about number of current players, and information about tournaments are also good ways to make the gaming experience more exciting.

Jackpots are unavailable when using play bonus money.

Note:
You may have a different integration set-up depending on your particular jurisdiction.

Display Jackpot Information

The following SOAP API methods provide jackpot information in a form, suitable for display in scrolling banners, tickers, and announcements:

Method	Returns
`getCurrentJackpot`	Information about the current jackpot size for a specific jackpot. Not all games have jackpots, and some games have multiple jackpots or pooled jackpots.
`getIndividualJackpotInfo`	Individual jackpot information, per game. Information includes Jackpot Name, Jackpot Type, Current Jackpot Value, Last Payout Date, Total Jackpot Amount Paid Out, and Jackpot Payout Count.

You can also use the named query 'GetJackpotsforGames' to return jackpots for all games.

Prevent Fraud and Irregularities in Jackpot Games

To ensure that every player has a fair chance of winning the jackpot, the system is designed to throttle jackpot games.

This means that if a player plays game rounds faster than twice per second, the spin requests will be queued up and processed only twice per second. The reason for this behaviour is to

eliminate the risk of abusive players winning small jackpots by flooding the game with spin requests once the jackpot is big enough.

All jackpot games are throttled by default.

Publish Jackpot Terms and Conditions

The standard game rules that NetEnt supplies for games with progressive jackpots specify that players should refer to the gaming site for the terms and conditions of the jackpot payout.

> **Note:**
> Please ensure that your site publishes information about the terms and conditions imposed on players in the case of a jackpot payout, such as non-payout in the case of suspected fraud.

For assistance when creating terms and conditions, contact our Customer Support: support@netent.com.

Display Game History

A common service our operators offer their players is the ability to list historical data about their latest game rounds. This helps build trust towards the operator and can reduce Customer Support enquiries.

A link is automatically added to the **Game Settings** menu in each of our games, enabling players to view the history of their recent game rounds:

By default, the game history *feature* is enabled. If you do not wish to display the game history feature, contact support@netent.com to disable this feature.

> **Note:**
> Game history *tracking* is not enabled by default. To enable the game history tracking, contact support@netent.com.

Our Customer Support team can also configure the following features of game history for you:

- The number of game rounds displayed (default: 10).
- The logo displayed on the history page.
- The display fonts and background colours.
- The format of date, time and currency displayed.
- The language used in the game tracking pages.

> **Note:**
> You may have a different integration set-up depending on your particular jurisdiction.

To add a link to the history page elsewhere on your site, use a URL as described in Customer Specific Environment URLs, with the following added parameters:

Parameter	Description
`lang`	A two-character ISO 639 language code in lower-case, for example `en`.
`sessionId`	The player's current session ID that was previously returned by `loginUser` or `loginUserDetailed`.
`gameRoundId`	The game round ID or 'last' to show the last played game round.
`gameId`	An optional parameter that is a unique identifier for a game, for example `roulette_sw`. ■ If <`gameId`> is sent and is valid, only that game's rounds display. ■ If <`gameId`> is sent and is invalid, no game rounds display. ■ If no <`gameId`> is sent, game rounds for all played games display. **Note:** For a list of valid game identifiers contact your Technical Integration Manager.
`backURL`	Used on the game rounds details page for the button back to the history page.

Note:
The game history can be referenced from the games or directly from the casino lobby by entering any of the history URLs.

Game History:

```
https://<game service url>/history?lang=<lang>&sessionId=<sessid>
```

Details:

```
https://<game service url>/detail?gameRoundId=<gameroundid>
&sessionId=<sessid>&gameId=<gameid>&lang=en&backURL
=%2Fgame%2Fhistory%3Flang%3Den%26sessionId%3D=<sessid>
```

Details Last Game Round:

```
https://<game service url>/-
detail?gameRoundId=last&sessionId=<sessid>
```

Note:
Players can view the tracking information for the last finished game round by providing the value `last` to the '`gameRoundId`' parameter in the URL. Unfinished game rounds are subject to the terms and conditions of the gaming site. For more information refer to the gaming website.

For more information, see Unfinished Game Rounds.

Unfinished Game Rounds

There are ways of displaying and directing the players to game rounds that they never completed, to allow the players to finish the game rounds.

Method	Description	Returns
`hasUserUnfinishedRound`	Determines if a player has an unfinished game round.	boolean
`List unfinished game rounds`	This method is a Named Query executed by the `executeNamedQueryMatrix` method.	`UserId`: Integer as defined by CasinoModule (not to be confused with `userName`). `gameRoundId`: ID of the game round. This is used in the `finishGameRounds` method to close the game round. `sessionId`: Player session ID. `startDate`: Game round start date. `gameId`: ID of the game. `PlayerBetAmount`: Amount of the bet.
`finishGameRounds`	Finishes unfinished game rounds specified in the `gameRoundId` parameter.	A list of objects with information about finished games and game rounds, including bets and wins.

Note:
There is also an automated job in CasinoModule to finish unfinished game rounds. This job is disabled by default but may be enabled by NetEntCustomer Support (support@netent.com). If the job is enabled, the default setting is to finish game rounds that have been open for 6 months.

Publish Unfinished Game Rounds and Session Timeouts Terms and Conditions

The standard game rules that NetEnt supplies for games specify that players should refer to the gaming site for the terms and conditions for handling of the unfinished game rounds and session timeouts.

Note:
There are methods that you can use to determine if users have unfinished game rounds and to finish unfinished game rounds for players. For more information, see *CasinoModule™ SOAP API* documentation.
Also, please ensure that your site publishes information about the terms and conditions imposed on players in the case of an unfinished game round, such as non-payout in the case of suspected fraud, and session timeouts.

Affiliate Programs

Affiliates draw traffic to a site and collect a commission based upon the revenue their players generate. Operators who run affiliate programs can use CasinoModule's ADMIN TOOL to generate statistics and reports about their affiliates.

Register Affiliate

Each affiliate must have a unique identification code.

> **Note:**
> The merchant account (the account used to access the SOAP API) is the default affiliate in the system. If no `AffiliateCode` is specified, then the player is tagged to the merchant account that was used in the method call.

To register a new affiliate with CasinoModule, call the `registerUser` method and pass the following fields in the parameter called `extra`:

Field	Description
`RegisterAffiliateCode`	The unique identification code for the new affiliate (max length: 16 characters).
`Comment`	A comment about the new affiliate.
`RoyaltyPercent`	The percentage of a player's revenue that the new affiliate should get (0 to 100).
`CostPerAcquisition`	The amount that the affiliate receives when a player deposits for the first time.

Alternatively,

- Use the **Create a new affiliate** area on the **Affiliate statistics** page in the **Statistics** menu.

or

- Call the `loginUserDetailed` or `registerUser` method and pass the affiliate code (that does not exist in the system till now) as a parameter. A new affiliate with the new code is created.

Affiliate User Name Prefix

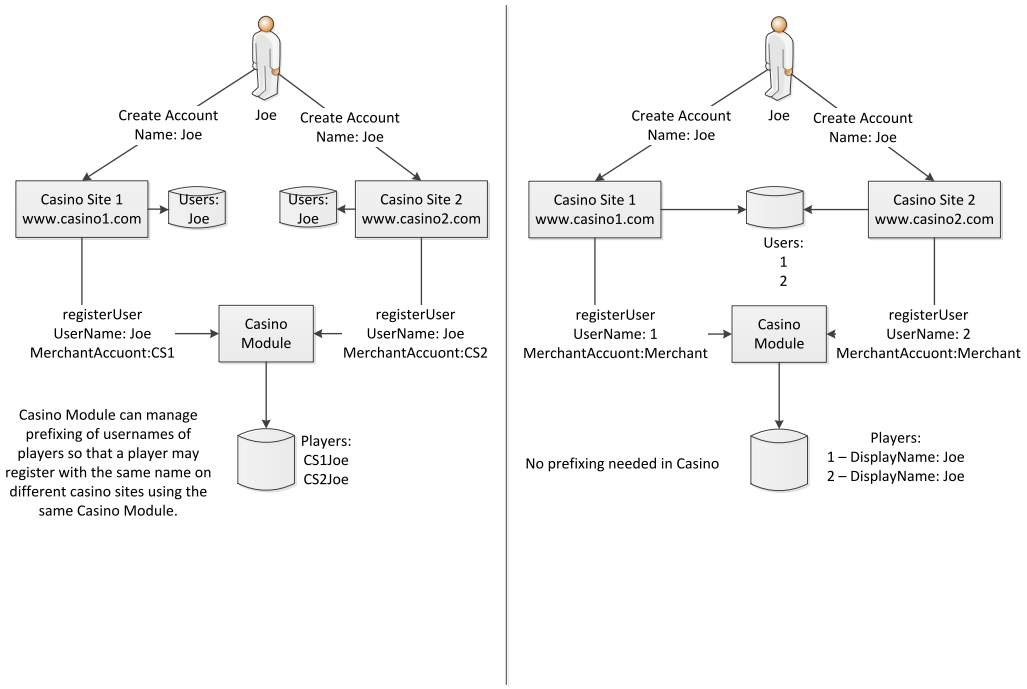
When creating a new affiliate by sending in an affiliateCode that does not exist to the methods `loginUserDetailed` or `registerUser`, the affiliates get the username as 'aff-' + <`AffiliateCode`>, for example aff-1. The affiliate will get the same password as the merchant account.

Player User Name Prefix

Some operators have several casino websites with separate player databases, but all these interact with the same CasinoModule. To be able to see from which casino site a player comes from, we recommend that you have several merchant accounts that include the different casino specific affiliate codes in the account name, and then add a merchant account prefix to the UserNames.

For example, two players with the same `userName` xyzPlayer (one from the casino CasinoSite1 and the other from the casino CasinoSite2) get the following user names in CasinoModule, CasinoSite1xyzPlayer and CasinoSite2xyzPlayer.

The image below illustrates an example for how to prefix user names.



Associate Player with Affiliate

To tag a player with an affiliate code, call the `loginUserDetailed` or `registerUser` method and pass the following field in the parameter called `extra`.

Note:

If the `AffiliateCode` extra parameter is not included in the call the player will be associated with the merchant account used in the call.

Field	Description
`AffiliateCode`	The unique identification code for an affiliate. If this `AffiliateCode` is unknown to CasinoModule, a new affiliate account will be opened. Then: ■ If the player currently has no `AffiliateCode` associated, CasinoModule will tag the player with this `AffiliateCode`. ■ If the player already has an `AffiliateCode` associated and this `AffiliateCode` is different, the player will be tagged with this new code.* ■ If the player already has an `AffiliateCode` associated and this `AffiliateCode` is not given, the existing affiliate tagging will be removed (the player will be tagged as belonging to the operator alone).*

* Changes to the affiliate codes will *not* be reflected in the historical affiliate statistics in ADMIN TOOL.

Multi-tier Affiliate System

CasinoModule supports a multi-tier affiliate system, with affiliates having sub-affiliates.

By default CasinoModule has one master merchant account with an empty affiliate code. When new affiliates are created, they will be sub-affiliates to the merchant account used in the call when creating the affiliate.

CasinoModule has support for multiple master affiliates and these can be set up by NetEnt by request to support@netent.com.

The affiliate created with the `RegisterAffiliateCode` parameter will be a sub-affiliate to the affiliate included in the `AffiliateCode` parameter. The `AffiliateCode` affiliate will be created if it does not exist and will be a sub-affiliate to `merchantId` affiliate.

Tournaments

This topic contains information on how to manage the Tournament Lobby.

Tournament Lobby Recommendations

When setting up your casino lobby, you should consider the following:

- Deploy a separate tournament lobby where players can view scheduled tournaments and see the ones they are eligible to attend.
- Divide the tournaments listed in the lobby into three categories: ongoing tournaments, future tournaments, and recently finished tournaments.
- In the tournament lobby, show only the information that is most relevant to a tournament such as: name of tournament, start time, end time.
- For each tournament listed in the lobby, link to a separate tournament details page containing more detailed information about the tournament, and displaying a link to enter the tournament.
- Present a list of previous winners and prizes.

Tournament Details Page Recommendations

When setting up your details page, you should consider the following:

- Deploy a specific tournament details page where players can view information about a tournament.
- Display the leaderboard with final result and winners.

Play Game in Tournament

The player enters the tournament by clicking a link to a game on the tournament details page or in the tournament lobby.

Note:

- The game link must send the `tournamentId` to the game page.
- For scoring calculations to work according to expectations, it is recommended not to facilitate playing on several tournament games simultaneously.

Tournament Methods in SOAP APIs

This topic lists the methods in the CasinoModule SOAP APIs that relate to tournaments, and gives a brief description of their use.

SOAP WS-I Compatible SOAP API

Method	Description	Returns
`getAllTournamentsV2`	Requests information on all ongoing and future tournaments within the given time period.	A list of tournaments in the specified time period.
`getTournamentDetailsV3`	Requests tournament occurrence details for a specific occurrence. This method provides all prizes in both player currency and casino currency, assuming that a player currency is provided. If no player currency is provided, prizes are listed in all available currencies.	An object representing the tournament occurrence.
`getTournamentResult`	Requests the result of the specified tournament occurrence as an array of tournament standings (the final state of the leaderboard).	The tournament result.
`getUserTournamentsV2`	Gets information on all tournaments a particular user is allowed to participate in.	A list of all tournaments the player can participate in.
`isUserInTournament`	Checks if the player has participated in the tournament.	Returns if the player can participate in the tournament.

Display List of Available Tournaments

When players enter the tournament lobby, they normally expect to see a list of all the tournaments in which they are allowed to participate. That is, where the players pass the tournament filters.

To get a list of the tournaments within a specified period for which the player is eligible, use the following method:

- `getUserTournamentsV2`

Note:
This method is also useful if you want to highlight a list of tournaments elsewhere on your website (for example, in a side-bar in the casino lobby).

Operators can configure the layout of the page so that players get an overview of, for example, ongoing, future and finished tournaments. With this kind of layout, players can easily keep

track of their past, current, and future tournament activities. Operators are of course free to exclude classic tournaments from the tournament lists.

Note:
The `tournamentId` should be passed to the tournament details page from the tournament lobby page.

Display Tournament Details

To get detailed information about a tournament, use the following method:

- `getTournamentDetailsV3`

You can also use the `getTime` method to get the correct time (that is, the current database time) to check if the tournament is ongoing, future, visible, etc.

Operators can configure the layout of the page to include information about, for example, start and end times, the playing currency, the current prize pot (which can grow if the prize type is 'share'), prize model, and so on.

Leaderboard

The leaderboard displays information about the success of players in a tournament. For browser-based tournament games, the leaderboard can be launched in either an HTML page or in a pop-up window.

Leaderboard Exclusion Threshold

The leaderboard has a threshold that excludes players who have played too few rounds from the display. This threshold is initially 10% of the tournament's minimum number of rounds. If at least one player has played the minimum number of rounds, the threshold is increased for subsequent players to 20% of the minimum number of rounds.

Example:
If the minimum number of rounds is 100, the lower threshold is set to 10 rounds at the start of the tournament. Only players who have played 10 or more rounds are shown on the leaderboard. As soon as one player has completed 100 game rounds, 20 game rounds are required for subsequent players to be displayed on the leaderboard.

Player's Name Display

The leaderboard displays the player’s display names. If the CasinoModule user profile contains the player's nickname, that nickname will be displayed on the leaderboard. If not, the player's user name will be displayed.

Publish the Leaderboard

The information from the `getUserTournaments` or `getUserClassicTournamentsV2` method can be used to decide when to launch the leaderboard for a tournament. The previously returned `tournamentId` is also used as parameter.

The leaderboard can be published and customized to your own preferences using HTML5, Javascript and CSS (Cascading Style Sheets).

Publish and Customize with HTML5, Javascript and CSS

This section describes how to publish and customize the tournament leaderboard.

Included at the end of this section you will find a tournament leaderboard widget example. The widget example comprises Javascript and CSS (Cascading Style Sheets) code. The code examples can be customized to your own preferences and company branding.

Follow the procedure below to publish and customize the leaderboard.

Step by Step

1. Use the following HTML template provided below to launch the leaderboard.

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
        body {
        background-color:#a0a0a0;
        }
        </style>
        <title>Game web page</title>
        <script type="text/javascript" src-
        ='http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js'></scrip-
        t>
        <script type="text/javascript" src="tournamentwidget.js"></script>
        <link type="text/css" rel="stylesheet" href="tournamentwidget.css" medi-
        a="all">
        <script type="text/javascript">
        $(document).ready(function() {
        var userName = "<username>";
        var gamesession = "<gamesession>";
        var tournamentId = <tournamentId>;
        var gameserver = "<gameserver>/servlet/InfoServlet";
        netent_tournamentwidget.initialize(username, gameserver, gamesession, tour-
        namentid);
        });
        </script>
    </head>
    <body>
        <div class='tournamentbody'>
        <div class='tournamenholder'>
        <span class='tournamentheader'>LEADERBOARD</span>
        </div>
        <div id="tournamentlist" class="tournamentlist"></div>
        </div>
    </body>
</html>
```

2. Modify the highlighted parameters as follows.

Attribute/Parameter	Description
`userName`	A logged in player's CasinoModule username.
`gameserver`	CasinoModule game server protocol and host. For example (https://server.casinomodule.com) appointed by NetEnt AM (Account Manager).
`gamesession`	A logged in player's CasinoModule game session.
`tournamentId`	The ID of the tournament. This can be obtained when setting up the tournament in ADMIN TOOL.

3. Modify the code in section Widget Javascript Code Example and Widget CSS Code Example as you see fit to apply your company branding to the leaderboard, with your designated logo and colors.

Widget Javascript Code Example

Table 1: Javascript Code Example

Example:

```
<!DOCTYPE html>
// Todo: Add timeout handling if request never returns
// Todo: Make parameters soft: width, height, server url, username, player, competition
id
// Todo?: Make it brandable?
$.support.cors = true;
var netent_tournamentwidget = {};
   netent_tournamentwidget.initialize = function(player, serviceUrl, sessionId, com-
   petitionId)
   {
      this.player = player;
      this.timeout = 10000;
      this.updateinterval = 10000;
      this.serviceUrl = serviceUrl;
      this.sessionId = sessionId;
      this.competitionId = competitionId;
      this.showNegativeScores = false;
      this.jsonobj = undefined;
      this.serverDate = undefined;
      this.HTTP_METHOD = "GET";
      this.errormsg = "";
      this.displayMessage("<br><div class='tournamentinfo'>Please wait...</div>");
      var self = this;
      this.errormsg = "Initialization failed.";
      this.refreshtimer = setInterval(function() { self.renderPresentation() }, 1000);
            $.ajax({
               cache:false,
               timeout:this.timeout,
               url: netent_tournamentwidget.serviceUrl + "?op-
               eration=toplistinit&competitionid=" + this.competitionId + "&username="
               + this.player + "&format=json&s=" + this.sessionId + "&no-cache=" +
               (Math.floor(Math.random() * 10000000)),
               type: self.HTTP_METHOD,
               success: function(data) {
                  self.showNegativeScores = data.TournamentInfo.ShowNegativeScores;
                  self.showCurrency = data.TournamentInfo.ShowCurrency;
                  if(data.TournamentInfo.NoOfDecimals!=undefined)
                  {
                  self.noOfDecimals = parseInt(data.TournamentInfo.NoOfDecimals);
                  } else
                  {
                  self.noOfDecimals = 2; // Default to 2 decimal display unless server
                  provides a specific value
                  }
                  self.minNoOfRounds = data.TournamentInfo.MinNoOfRounds;
                  self.currencySymbol = data.TournamentInfo.CurrencySymbol;
                  self.tournamentName = data.TournamentInfo.Name;
                  self.updateFromServer();
                  self.intervaltimer = setInterval(function() { self.updateFromServer()
                  }, self.updateinterval)
               },
               error: function(xhr, textStatus, e) {
                  clearInterval(self.intervaltimer);
                  clearInterval(self.refreshtimer)
                  self.errormsg = self.getErrorMessage(xhr.status);
                  self.displayMessage("<br><div class='tournamenterrorheader'>An error
                  occured:</div><div class='tournamenterrortext'>" + self.errormsg +
                  "</div>");
                  console.log(self.getErrorMessage(xhr.status));
               }
            });
         }
```

Table 1: Javascript Code Example (Continued)

Example:

```
netent_tournamentwidget.renderPresentation = function()
{
    if(this.jsonobj == undefined)
    {
        return;
    }
    var players = this.jsonobj.TournamentStanding.Player;
    if(this.currentServerTime != undefined)
    {
        this.currentServerTime = new Date(this.currentServerTime.getTime()+ 1000);
    }
    var tt = new Date(this.jsonobj.TournamentStanding.Time.replace("CET ", ""));
    var ts = new Date(this.jsonobj.TournamentStanding.TournamentStart);
    var te = new Date(this.jsonobj.TournamentStanding.TournamentEnd);
    var diff = Math.floor((te-tt) / 1000);
    var durationToStart = this.calculateTimeDiff(this.currentServerTime, ts);
    var durationToEnd = this.calculateTimeDiff(this.currentServerTime, te);
    var duration = this.calculateTimeDiff(ts, te);

    if (this.jsonobj.TournamentStanding.IsFinished && players.length>0) {
        console.log("Tournament finished.");
        console.log(this.jsonobj);
        clearInterval(this.intervaltimer);
        clearInterval(self.refreshtimer)
        var presentation = "<div class='tournamentname'>" + this.tournamentName +
        "</div>";
        presentation += "<br><div class='tournamentinfo'>Final results:</div>"
        presentation += "<div class='tournamentrow'>"
        presentation += "<div class='tournamentfield usernamelistheader'>Player</div>"
        presentation += "<div class='tournamentfield scorelistheader'>Score</div>"
        presentation += "</div>"
        var username = "";
        var rank = "";
        var score = 0;

        for(var j=0;j<players.length;j++)
        {
            if(players[j].UserName==netent_tournamentwidget.player)
            {
                username = players[j].UserName;
                score = players[j].Score;
                rounds = players[j].NoOfRounds;
                rank = players[j].Position;
                scorestr = parseFloat(score).toFixed(this.noOfDecimals);
                var qualified = false;
                if(rounds >= this.minNoOfRounds)
                {
                    qualified = true;
                }
                presentation += "<div class='tournamentrow'>"
                if(qualified)
                {
                    presentation += "<div class='tournamentfield username qualified'>" +
                    username + "</div>";
                    presentation += "<div class='tournamentfield score qualified'>" +
                    scorestr + "</div>";
                } else
                {
                    presentation += "<div class='tournamentfield username notqualified'>"
                    + username + "</div>";
                    presentation += "<div class='tournamentfield score notqualified'>" +
                    scorestr + "</div>";
                }
                presentation += "</div>";
                if(rounds>=0)
                {
                    if(qualified)
                    {
                        presentation += "<br><div class='qualified tournamentrounds'>"
```

Table 1: Javascript Code Example (Continued)

Example:

```
                    } else
                    {
                        presentation += "<br><div class='notqualified tournamentrounds'>"
                    }
                    presentation += "You have played " + rounds + " rounds."
                    presentation += "</div>"
                    if(qualified)
                    {
                        presentation += "<br><div class='tournamentinfo'>You finished: " +
                        rank + "</div>";
                    }
                }
            }
        }
        var e = document.getElementById("tournamentlist")
        e.innerHTML=presentation;
        return;
    }
    if(durationToEnd.inTotalSeconds < 0 && this.jsonob-
    j.TournamentStanding.IsFinished==false)
    {
        presentation = "<br><div class-
        ='tournamentinfo'>The tournament is closing...<br><br>Awaiting
        final results</div>"
        //clearInterval(this.intervaltimer);
        //clearInterval(this.refreshtimer);
        //console.log("Diff was negative: close tournament, disable polling.");
    } else {
        if(durationToStart.inTotalSeconds>0)
        {
        var presentation = "<div class='tournamentname'>" + this.tournamentName +
        "</div>";
        presentation += "<br><div class='tournamentinfo'>Tournament starts in<br><div
        class='tournamenttimeinfo'>" + this.getDurationString(durationToStart, true,
        true, true, true, true) + ".</div></div>";
        } else
        {
            var tlstr = this.getDurationString(durationToEnd, true, true, true, true,
            true);
            var presentation = "<div class='tournamentname'>" + this.tournamentName +
            "</div>";
            presentation += "<br>"
            presentation += "<div class='tournamentinfo'>Time left:<br><div class-
            ='tournamenttimeinfo'>" + tlstr + "</div></div>";

            if(players.length>0)
            {
                presentation += "<div class='tournamentrow'>";
                presentation += "<div class='tournamentfield user-
                namelistheader'>Player</div>";
                presentation += "<div class='tournamentfield sco-
                relistheader'>Score</div>";
                presentation += "</div>";
                var rounds = 0;
                var playerpresentation = "";

                for(var j=0;j<players.length;j++)
                {
                    presentation += "<div class='tournamentrow'>"
                    var username = players[j].UserName
                    var score = players[j].Score;
                    scorestr = parseFloat(score).toFixed(this.noOfDecimals);
                    if(parseFloat(score)<0 && this.showNegativeScores==false)
                    {
                        scorestr = "&lt" + parseFloat("0").toFixed(this.noOfDecimals);
                    }
                    if(players[j].UserName == netent_tournamentwidget.player)
                    {
```

Table 1: Javascript Code Example (Continued)

Example:

```
                    rounds = players[j].NoOfRounds;
                }
                var qualified = false;
                if(rounds >= this.minNoOfRounds)
                {
                    qualified = true;
                }

                if(netent_tournamentwidget.player==username)
                {
                    //is Player Qualified?
                    if(qualified)
                    {
                    playerpresentation = "<div class='tournamentfield username qual-
                    ified'><i>" + username + "</i></div>";
                    playerpresentation += "<div class='tournamentfield score qual-
                    ified'><i>" + scorestr + "</i></div>";
                    } else
                    {
                    playerpresentation = "<div class='tournamentfield username notqual-
                    ified'><i>" + username + "</i></div>";
                    playerpresentation += "<div class='tournamentfield score notqual-
                    ified'><i>" + scorestr + "</i></div>";
                    }
                    if(j < 10) //Only present top 10 players
                    {
                    presentation += playerpresentation;
                    playerpresentation = "";
                    }
                } else
                {
                    if(j < 10)
                    {
                        presentation += "<div class='tournamentfield username dimmed-
                        row'>" + username + "</div>";
                        presentation += "<div class='tournamentfield score dimmed-
                        row'>" + scorestr + "</div>";
                    }
                }
                presentation += "</div>"
            }

            if(playerpresentation!="")
            {
            presentation += "<div class='tournamentrow outsidelist'>" + play-
            erpresentation + "</div>"
            }
        }
        if(rounds>=0)
        {
            if(qualified)
            {
            presentation += "<br><div class='qualified tournamentrounds'>"
            } else
            {
            presentation += "<br><div class='notqualified tournamentrounds'>"
            }
            presentation += "You have played " + rounds + " rounds."
            presentation += "</div>"
        }
    }
}
var e = document.getElementById("tournamentlist")
e.innerHTML=presentation;
}

netent_tournamentwidget.displayMessage = function(msg)
{
    var e = document.getElementById("tournamentlist");
```

Table 1: Javascript Code Example (Continued)

Example:

```
        if(e!=undefined)
        { e.innerHTML = msg; }
    }
    netent_tournamentwidget.getErrorMessage = function(errorCode)
    {
        switch(errorCode) {
        case 404 : return "Page not found";
        break;
        case 460 : return "Unknown operation";
        break;
        case 461 : return "Unknown user/login";
        break;
        case 462 : return "Unknown tournament id";
        break;
        case 500 : return "Server error";
        break;
        case 560 : return "Unknown server error";
        break;
        case 561 : return "Game session timeout<br>Please reload the page.";
        break;
        }
    }

    netent_tournamentwidget.updateFromServer = function()
    {
        try {
            var self = this;
            var rounds = undefined;
            $.ajax({
                cache:false,
                timeout:this.timeout,
                url: netent_tournamentwidget.serviceUrl + "?op-
                eration=toplist&competitionid=" + this.competitionId + "&username=" + this.-
                player + "&format=json&s=" + this.sessionId + "&no-cache=" + (Math.floor
                (Math.random() * 10000000)),
                type: self.HTTP_METHOD,
                success: function(data){
                    if(data.length==0)
                    {
                        return // Ignore empty responses.
                    }
                    self.jsonobj = data;
                    var currentServerTime = new Date(self.jsonob-
                    j.TournamentStanding.Time.replace("CET ", ""));
                    self.currentServerTime = currentServerTime;
                },
                error: function(xhr, textStatus, e) {
                    clearInterval(self.intervaltimer);
                    clearInterval(self.refreshtimer)
                    self.displayMessage("<br><div class='tournamenterrorheader'>An error
                    occured:</div><div class='tournamenterrortext'>" + self.errormsg +
                    "</div>");
                    console.log(self.getErrorMessage(xhr.status));
                }
            });

        }
        catch (e) {
            clearInterval(self.intervaltimer);
            clearInterval(self.refreshtimer);
            self.displayMessage("<br><div class='tournamenterrorheader'>An error occured:</-
            div><div class='tournamenterrortext'>" + self.errormsg + "</div>");
            console.log(e.message);
        }
    }

    netent_tournamentwidget.calculateTimeDiff = function(datetime1, datetime2)
    {
        var diff= {};
```

Table 1: Javascript Code Example (Continued)

Example:

```
        var tt = new Date(datetime1);
        var te = new Date(datetime2);
        diff.inTotalSeconds = Math.floor((te-tt) / 1000);
        diff.inWeeks = Math.floor(diff.inTotalSeconds/((3600*24)*7));
        diff.inDays = Math.floor((diff.inTotalSeconds - (diff.inWeeks * (3600*24)*7)) /
        ((3600*24)));
        diff.inHours = Math.floor((diff.inTotalSeconds - (diff.inWeeks * (3600*24)*7) -
        (diff.inDays * (3600*24))) /3600);
        diff.inMinutes = Math.floor((diff.inTotalSeconds - (diff.inWeeks * (3600*24)*7) -
        (diff.inDays * (3600*24)) - (diff.inHours * 3600)) / 60)
        diff.inSeconds = Math.floor((diff.inTotalSeconds - (diff.inWeeks * (3600*24)*7) -
        (diff.inDays * (3600*24)) - (diff.inHours * 3600) - (diff.inMinutes * 60)))
        return diff;
    }

    netent_tournamentwidget.getDurationString = function(duration, includeWeeks,
    includeDays, includeHours, includeMinutes, includeSeconds)
    {
        var tlarr = [];
        if(duration.inWeeks>0 && includeWeeks)
        {
            var suffix = (duration.inWeeks > 1 ? "s" : "");
            tlarr.push(duration.inWeeks + " week" + suffix);
        }
        if(duration.inDays>0 && includeDays)
        {
            var suffix = (duration.inDays > 1 ? "s" : "");
            tlarr.push(duration.inDays + " day" + suffix);
        }
        if(duration.inHours>0 && includeHours)
        {
            var suffix = (duration.inHours > 1 ? "s" : "");
            tlarr.push(duration.inHours + " hour" + suffix);
        }
        if(duration.inMinutes>0 && includeMinutes)
        {
            var suffix = (duration.inMinutes > 1 ? "s" : "");
            tlarr.push(duration.inMinutes + " minute" + suffix);
        }
        if(duration.inSeconds>0 && includeSeconds)
        {
            var suffix = (duration.inSeconds > 1 ? "s" : "");
            tlarr.push(duration.inSeconds + " second" + suffix);
        }
        var tlstr = tlarr.join(", ");
        tlstr = tlstr.slice(0, tlstr.lastIndexOf(", ")) + tlstr.slice(tlstr.lastIndexOf(",
        ")).replace(", ", " and ");
        return tlstr;
    }
```

Widget CSS Code Example

Table 1: CSS Code Example

Example:

```
.tournamentbody {
	background-color:#c0c0c0;
	color:#000000;
	width:180px;
	display:inline-block;
	border-radius:10px;
	border:#c0c0c0 solid 2px;
	overflow:none;
}
.tournamenholder {
	text-align:center;
}
.tournamentheader {
	color:#202020;
	font-family:Arial;
	font-size:20px;
	display:inline;
}
.tournamentname {
	color:#ffffff;
	font-family:Arial;
	font-weight:bold;
	font-size:15px;
	display:inline-block;
	background-color:#222222;
	padding:2px 2px 2px 2px;
	margin: 0px 0px 10px 0px;
	width:176px;
}
.tournamentinfo {
	font-family:Arial;
	font-size:12px;
	display:block;
	color:#ffffff;
	width:180px;
	margin-bottom:10px
}
.tournamenttimeinfo {
	font-family:Arial;
	font-size:10px;
	display:block;
	color:#ffffff;
	width:180px;
}
.tournamentrow {
	display:block;
	color:#ffffff;
	width:180px;
	font-size:12px
}
.outsidelist {
	padding-top:8px
}
.tournamentlist
{
	text-align:center;
	background-color:#000000;
	border-bottom-left-radius:10px;
	border-bottom-right-radius:10px;
	height:320px;
}
```

Table 1: CSS Code Example (Continued)

Example:

```
.score
{
	font-weight:normal;
	width:80px;
	font-family:Arial;
	text-align:right;
}
.tournamentfield
{
	display:inline-block;
	text-align:left;
}
.usernamelistheader
{
	color:#ffffff;
	font-weight:bold;
	font-family:Arial;
	width:90px;
	text-align:left;
}
.username
{
	color:#ffffff;
	font-weight:normal;
	width:90px;
	font-family:Arial;
	text-align:left;
}
.scorelistheader
{
	color:#ffffff;
	font-weight:bold;
	font-family:Arial;
	width:80px;
	text-align:right;
}
.score
{
	font-weight:normal;
	width:80px;
	font-family:Arial;
	text-align:right;
}
.dimmedrow {
	color:#a0a0a0;
}
.fieldselected
{
	color:#00ff00;
}
.qualified
{
	color:#00ff00;
}
.notqualified
{
	color:#009933;
}
.tournamentrounds
{
	font-family:Arial;
	font-size:12px;
	display:block;
}
.tournamenterrorheader {
	color:#ff0000;
	font-weight:bold;
	font-family:Arial;
	font-size:14px;
```

Table 1: CSS Code Example (Continued)

Example:

```
}
.tournamenterrortext {
	color:#ff0000;
	font-weight:normal;
	font-family:Arial;
	font-size:12px;
}
```

Extended Tournament Support

To increase the common interest for tournaments, you can query CasinoModule for other tournament related information and display it on your site. The following multi-currency SOAP API methods provide more information about tournaments:

Method	Description	Returns
`getCurrentOverview`	Provides current statistics ready for presentation in a scroller or ticker.	Number of players, jackpot info for each game, most popular game ID, winner's user name, winner's country, winning amount, information about the 3 last big wins.
`getLeaderBoard`	Provides leaderboard details for a specific tournament. The leaderboard contains the top N players (N typically has the value of e.g. 10) with the highest score in the tournament. The score depends on the tournament win criteria.	A list of tournament standings, representing player positions and results.

Bonus Programs

This topic describes how CasinoModule bonus programs affect the technical integration of CasinoModule with an operator site.

Wallet Options and Bonus Programs

The type of wallet options chosen for the implementation affects the types of bonus programs available:

- For Basic wallet implementations, all bonus program types are supported.
- For Seamless wallet games, deposit bonus is not supported.

The type of wallet implementation chosen determines how withdrawals from CasinoModule accounts are handled.

Withdraw Money and Deposit Bonus Programs

Deposit bonus programs complicate withdrawals from the accounts of players that have received bonuses. Until the wagering requirements are fulfilled, it should not be possible for the player to withdraw any bonus money (including the bonus money winnings) from his account. Furthermore, if a player selects to withdraw real money from his account, he risks

losing what has been previously added to his account as bonus money. If the player chooses to forfeit the bonus money and makes a withdrawal, the remaining bonus money disappears. If the player tries to withdraw his entire account balance, the withdrawal will fail.

Example: If a player makes a deposit of $100 and gets a $10 bonus, he will have a total balance of $110. If he tries to make an immediate withdrawal of $110, it will fail because of insufficient funds in his account. (What actually happens is that the bonus money is removed, leaving the balance at $100; after that, a withdrawal is made but fails because the player does not have the requested $110.)

For games with the wagering contribution factor set to 0, you can request that display and usage of bonus money is blocked by contacting support@netent.com.

The getUserBonusInfoV4 Method

To withdraw money from a player's account when deposit bonus programs are active, first call `getUserBonusInfoV4` to get the user's current account balance and bonus money information. This method returns the player's account balance and bonus money information from all programs in which the player is participating as an array of key/value strings.

Note:
For your particular jurisdiction you may have a different integration setup. Therefore, please see separate documentation for your jurisdiction.

The values are floating-point monetary amounts in the casino currency, as follows:

Key/value String	Description
`accountBalance`	The player's account balance, including any bonus money on which there are outstanding wagering requirements.
`totalBonus`	The total amount of bonus money awarded to the player on which there are still outstanding wagering requirements. This is the amount that the player will forfeit if he opts to withdraw funds from his account. This amount can be greater than `accountBalance`, but `accountBalance` can never be negative.
`totalRequirement`	The remaining wagering requirement, that is the amount a player still has to bet to fulfill the **total wagering requirement** for the bonus money.
`wageredAmount`	The total amount wagered so far in the outstanding bonus programs. Add this amount to the `totalRequirement` (that is the remaining wagering requirement) amount to calculate the **total wagering requirement** for the bonus money.
`totalCash`	The maximum amount that the player can withdraw without losing any of their bonus.
`totalSecurity`	The amount that is linked to bonuses. If the player withdraws any of this, some or all of his bonuses will be lost.

Note:
If all bonus requirements are fulfilled, only `accountBalance` will be returned.

Deposit Bonus Scenarios

There are three scenarios that may be encountered with deposit bonus programs. The following examples assume a 10% deposit bonus and a x10 wagering requirement.

Scenario 1	The player has not received a bonus, or has received a bonus earlier and either met the wagering requirements or made an early withdrawal.	`accountBalance`= 100.00 `totalBonus`= 0 `totalRequirement`= 0 `wageredAmount`= 0
Scenario 2	The player has received a $10 bonus for a $100 deposit and has not started playing after receiving the bonus.	`accountBalance`= 110.00 `totalBonus`= 10.00 `totalRequirement`= 100.00 `wageredAmount`= 0
Scenario 3	The player has received a $10 bonus for a $100 deposit, has started playing and placed bets for $40, with a loss of $30 so far. The player needs to bet an additional $40 before he can make a full withdrawal.	`accountBalance`= 80.00 `totalBonus`= 10.00 `totalRequirement`= 40.00 `wageredAmount`= 40.00

Inform Players about Bonus Withdrawal Restrictions

We strongly recommend that you inform your players of the withdrawal restrictions on bonuses close to where they make withdrawals on the cashier page. You can add an extra field showing if the player has received a bonus close to where the current balance is displayed, thus:

Balance: $110
(Amount available for withdrawal: $100; Bonus not yet qualified for: $10)

If the player attempts to make a withdrawal before the wagering requirement has been met, display an alert informing the player that the bonus (including the bonus money winnings) will be removed from the account if the withdrawal is made. The player can then choose to make the withdrawal anyway.

Note:
The player should be prevented from removing an amount that exceeds `accountBalance` minus `totalBonus`.

If required, detailed information about outstanding bonus awards, wagering requirements, etc., *per bonus program*, can be obtained by calling the `getUserBonusInfoDetailV5` method.

Free Rounds Bonus Programs

Free rounds bonus programs enable players to try out specific games without having to pay. This is not the same as playing a game in Play for fun mode as there is still a cost involved in terms of whether the player receives any winnings as a bonus or real money at the end of the free rounds.

getUserFreeRoundGames

The `getUserFreeRoundGames` method can be used to display games in which the user has free rounds. Free rounds are initiated at the time when a player logs in to CasinoModule so this method should only be called after player login.

giveFreeRoundsV2

The `giveFreeRoundsV2` method can be used to give a player or players free rounds immediately, without specifying a bonus program in advance. Until the free rounds have been played through, `forfeitFreeRounds` can be used to revert the offer for a player or players. This version of the method returns a list with success, and error in case of non registered players or other failure. It also returns 'coin levels' in addition to 'bet levels', and the 'bonusWallet' parameter is removed.

> **Note:** In order for the `giveFreeRoundsV2` API method and Give Player Free Rounds function to work as intended, it is required for the operator to implement the wallet service in compliance with the latest provided WSDL file.

forfeitFreeRounds

The `forfeitFreeRounds` method makes it possible to forfeit free rounds that were issued using `giveFreeRounds`. If the free rounds have been played through, any winnings will not be touched by this call.

getUserFreeRoundInformation

The `getUserFreeRoundInformation` method makes it possible to display all the free rounds available for the player, both the number of remaining active free rounds as well as pending free rounds.

Bonus Program Promotion Codes

You can grant bonuses per player based on promotion codes. You can select the promotion code from an existing list of codes or provide a new code. If the code is used for another active or future bonus program, then an alert message is displayed.

If a promotion code is configured for a bonus program, then players must enter the code; otherwise, the bonus is not granted. This is done through the SOAP API or the **Player profile** page in ADMIN TOOL. Players can also activate the code themselves while they register or during log on at your website.

The code then becomes inactive when a player uses it. You can reactivate the code (passing the same promotion code again) for recurring bonus programs such as Login bonus and Deposit bonus. One activation code may be mapped to multiple bonus programs.

Note:

Promotion codes are not case-sensitive and the maximum length is 50 characters. Special characters and spaces are not allowed.

To activate a bonus program using a promotion code, call the `activateBonusProgramForPlayer` method. This method activates bonus programs that have a promotion code for a certain player.

The bonus program cannot be activated until the next player session, that is, the next login call. The `activateBonusProgramForPlayer` method returns `bonusProgramId`'s, the status of activation (successful or unsuccessful) and the reason of failure.

Using the procedure above will display an alert notification if:

- No active bonus program is associated with a player for the provided activation code.
- The bonus program for the provided activation code is already active.
- The player is excluded from the bonus programs.

Frequent Player Points (FPP) Program

The frequent player points (FPP) program awards players points for each bet they make in CasinoModule games. The points should be withdrawn from CasinoModule and converted into money or merchandise on a regular basis to provide an incentive for players to play more and return to claim the money or merchandise that their FPP programs have generated. Research shows that such programs encourage players to bet more, stay longer, and return more often.

The FPP program can be used as a standalone program or integrated with existing points programs. In either case, the operator is responsible for creating the parts of the FPP program that convert points to money or merchandise. CasinoModule provides only the mechanisms for collecting and withdrawing points.

Note:

The CasinoModule FPP program is disabled by default. To enable the program, please contact support@netent.com.

FPP Program Functionality

When active, the FPP program awards points every time a player places a bet. The program can be accessed through both the SOAP API and through ADMIN TOOL.

SOAP API Method	Description
`depositUserFrequentPlayerPoints`	Deposit frequent player points to a player's CasinoModule account.
`getAllFrequentPlayerPoints`	Get a list of point balances for all players.
`getUserFrequentPlayerPointsBalance`	Get the points balance for an individual player.
`withdrawUserFrequentPlayerPoints`	Withdraw points from an individual player's account.
`getFrequentPlayerPointsTransactions`	Returns a list of all transactions (withdraws and deposits) for an individual player.

ADMIN TOOL Option	Description
Admin > Frequent player points	Configure the points awarded as a % of the bet per game.
Tracking > Player profile	See the amount of points accumulated by a player.
Tracking > Player profile > Add or remove points	Manually add or remove points from a player's account.
Statistics > Accounting	See a summary of all points held in CasinoModule accounts.

Configure FPP Points Awards

The amount of points to award per bet per game can be viewed and configured via the **Frequent player points** option on the CasinoModule ADMIN TOOL **Admin** menu. The amount is specified as a percentage of casino's currency unit. The percentage can be set at different levels for different games.

The default configuration can be modified to comply with an existing points program, or to meet your preferred program setup, for example to commend skill games, or to promote specific games.

Recommended FPP Program Integration Process

Follow the procedure below to implement an FPP program.

Step by Step

1. Review the default FPP award percentages, via ADMIN TOOL. Modify the percentages, if necessary.
2. Fetch the player's points balance using the `getUserFrequentPlayerPointsBalance` SOAP API method, and then add it to the display of the player's accumulated points on your site.
3. Set up an automatic job to withdraw all frequent player points from CasinoModule on a regular basis:
   - Fetch a list of all players and their points balance using the `getAllFrequentPlayerPoints` method.
   - Withdraw points from individual players using the `withdrawUserFrequentPlayerPoints` method.
   For different ways of accomplishing this, see the table below.
4. Convert player points to money or merchandise on a regular (daily/weekly/monthly) basis.

Depending on how regularly CasinoModule games are played on your site and on whether or not you already operate you own points program, step 3 can be done in one of the following three ways:

Situation	Recommendations
Busy site with existing points program	Withdraw points as often as possible, for example, once an hour. Convert points to money or merchandise in accordance with your existing points program.
Busy site without existing points program	Withdraw points, convert to money or merchandise, and then add to player accounts at the period that suits you. However, performing this operation at too long an interval, such as once a month, may cause performance degradation. To decrease server load, an option is to fetch and withdraw points on a daily basis (using a nightly job), and then to convert the points to money or merchandise with suitable periodicity (for example, once a month).
Less frequently played site without existing points program	Withdraw points at longer intervals, such as once a month.

Manage Play and Block Limits

A play limit is a restriction that can be set by the players themselves to restrict their own play. Examples include a limit on the maximum length of a playing session or on the maximum amount that can be bet within a certain period.

In addition to these types of limit, players can set a block limit which prevents them from playing at all until it expires.

Note:
Casino Managers can also set play limits and block limits on behalf of the player through CasinoModule ADMIN TOOL.

You can use the following SOAP API methods to manage player limits and block limits from your system:

Method	Description
`getPlayerLimits`	Gets a list of the current and any future limits on a player.
`removePlayerLimit`	Removes a specific play limit or block limit: ■ If the limit is historical, no action is taken. ■ If the limit is active, it will be set to expire in a week. ■ If the limit is in the future, it will be deleted.
`setPlayerBlockLimitEndDate`	A block limit is a restriction that the player can configure, to prevent from too much playing.
`setPlayerLimit3`	A player limit is a restriction the player can configure to prevent from too much playing, like the maximum loss per session or the maximum session length. ■ If no current limit exists, or the current limit is more

Method	Description
	lenient than the new limit, the new limit will be valid from the player's next login. ■ If a current harsher limit exists, the new limit will be valid in a week from now. ■ If a current limit exists and the `playerLimitId` is included in the call, the changes will be valid immediately. **Note:** If you use this method, you have to take into consideration any legal requirements for player limit change delays.

Currency Exchange Rates

In a multi-currency casino one of two methods is used to manage currency exchange rates between the casino currency and each player currency. During casino setup, NetEnt configures which method will apply to each of the available player currencies.

Manage Currency Exchange Rates

In a multi-currency casino where the exchange rate method for a player currency is set to Automated from Operator (Web service), you must use the `setCurrencyExchangeRate` method to automatically manage exchange rates.

Important:
It is important to update all rates with the same frequency. Do not update some currencies daily and other currencies once a week.

To set a new currency exchange rate for a player currency, call the `setCurrencyExchangeRate` method and pass the following attributes/parameters:

Attribute/Parameter	Description
`currencyISOCode`	The currency ISO code.
`currencyRate`	The exchange rate (Player Currency to Casino Currency) for the specified ISO code.
`fromDate`	The date from which the rate is valid. **Note:** If the from date is omitted, the rate takes immediate effect.
`merchantId`	The operator's merchant user name, as provided by NetEnt.
`merchantPassword`	The operator's merchant password, as provided by NetEnt.

Note:
All exchange rates use six decimals; however currencies are rounded to two decimal places when displayed to the player.

Important:
Exchange rate in SEK is not supported for player currency in GEL (Georgian Lari) and PEN (Peruvian Nuevo Sol) and must therefore be disabled for the operator.

Player Currency Conversion

After migrating from a single currency casino to a multi-currency casino you can convert a player to a different currency, if required.

When you convert a player to a new currency the following occurs:

- The player's single currency casino account status is set to inactive, and the old player is renamed to <username>convertedTo<CurrencyCode>. You can use ADMIN TOOL to retrieve information about the old player account.
- A new player with the same user name and password is created together with an empty Basic wallet account.
- Basic player details are transferred to the new player account.
- The current player limits are converted to the new currency using the currently available exchange rate rounded down to integer. Expired player limits (valid until past dates) are not converted.
- Any unfinished game rounds are completed, residual bets are paid out to the old account, and any game reservations are resolved.
- Confirmation information about the new user ID and the old account's balance and remaining unresolved bonuses is returned.
- The player's bonuses are forfeited and reported in the casino result.

Convert Player to new Player Currency

The process to convert a player to a new player currency is a multi-step process.

Note:
The API methods listed below are fully described in the CasinoModule™ SOAP API documentation on the **Client Area** of the NetEnt website.

Follow the procedure below to convert the player to a new player currency.

Step by Step

1. Convert the player to the new player currency using the `convertPlayerToNewCurrency` method. Pass the following attributes/parameters:

Attribute/Parameter	Description
`userName`	The user's login name.
`merchantId`	The operator's merchant user name, as provided by NetEnt.
`merchantPassword`	The operator's merchant password, as provided by NetEnt.
`currency`	The ISO code for the currency that the player will be converted to.

2. Withdraw the player's money from the old account using the `withdrawUserMoneyV3` method and deposit the amount in player currency into the new account using the `depositUserMoneyV3` method (see Deposits).
3. As all bonuses are forfeited when performing the conversion, compensate the player for lost bonuses using one of the following:
   - Direct remuneration, using the `depositUserMoneyV3` method.
   - Set up new bonuses that correspond to the old bonuses using the `giveManualBonus` method.
   - A combination of `depositUserMoneyV3` and `giveUserManualBonusV5`.

Note:
Please contact NetEntCustomer Support for assistance with tasks associated with conversion of players to a new player currency.

Integration Reference

APIs provide the main integration points for your casinos. Methods in these APIs are used to manage a wide range of casino functions such as player account and currency options, player registration and login, cashier functions such as deposits and withdrawals, tournaments and bonus programs, as well as a number of casino reporting options.

The APIs you use depend on the wallet options and currency options you will have in your casino operations, and the type of operator you are (that is, a new or existing operator).

This section provides key information about the APIs used for integration, and describes the documents and resources available to assist in integration with CasinoModule.

SOAP API

NetEnt provides a public SOAP API, CasinoModule SOAP WS-I Compatible SOAP API, that is a standard web service interface defined via WSDL.

WSDL Files

NetEnt provides a machine-readable WSDL description in XML for each of the CasinoModule SOAP APIs. Depending on the tool you use on your side for web service communications, you will need to do one of the following:

- For Java and .NET, create stub code based on the WSDL file.
- For scripting languages like PHP and ASP, have your application parse the WSDL file. (Start by downloading a copy of the WSDL file and store it locally. Do not download the file on each page execution.)

The SOAP API WSDL files are located in both the production and generic test environments.

Important:
You must replace the end point URL specified in the WSDL file with the URL provided by NetEnt. Always replace http with https.

SOAP API Documentation

NetEnt publishes detailed information for each of the CasinoModule SOAP APIs in HTML format. This documentation contains a summary of the API methods, guidelines for using

those methods and caching responses, exception handling and error codes, and a list of the methods required for minimal integration.

The API Documentation is published as javadoc files at the following locations:

- **Client Area** of the NetEnt website
- Production/generic test environments

Operator and Merchant Identification

All sensitive calls to CasinoModule require the operator's system to provide a `merchantId` and `merchantPassword`. Unless you are intending to implement a two-tier affiliate program, you will require only one `merchantId` and `merchantPassword` that can be used in all SOAP API calls. These are supplied to you by your Technical Integration Manager (integration@netent.com).

Response Caching

For performance reasons, cache response data for as long as possible to reduce the number of calls you need to make to this service.

Response Formats

The ordering of elements is not fixed and can vary from call to call. You must always check that you are reading the correct element (that is, you **must not** rely on the order of elements). For instance, the order of the key-value pairs from Array methods is not fixed.

Handle Exceptions

Exception handling and error codes are described in the *CasinoModule™ SOAP API* documentation available on the **Client Area** of the NetEnt website.

The operator is responsible for making sure that all exceptions thrown by the CasinoModule SOAP API are handled in the appropriate manner.

> **Note:**
> Different tools for developing SOAP clients may handle SOAP exceptions in different ways, and some tools do not support SOAP exceptions at all.

For advice and guidance on selecting appropriate SOAP client development tools for your environment, email our Customer Support team: support@netent.com.

Appendix: Migrate to WS-I Compatible SOAP API

This table illustrates the migration from the previous multi-currency APIs to CasinoModule WS-I Compatible SOAP API, that is, the following information:

- Removed methods and what methods to use instead of the removed ones
- Parameter changes
- Unchanged methods

Methods in old API	Methods in new API	Action taken	Comment
ping	`ping`	No change	
registerTestUser	`registerTestUser`	Removed parameter from request: password	
registerUser	`registerUser`	Removed parameter from request: password	
registerUserWithPromotionCode		Removed	The combination of activateBonusProgramForPlayer and registerUser should be used instead
loginUserWithPromotionCode		Removed	The combination of activateBonusProgramForPlayer and loginUser should be used instead.
loginUser		Removed parameter from request: password Added optional parameter in request: currencyISOCode	
loginUserDetailedWithPromotionCode		Removed	The combination of activateBonusProgramForPlayer and loginUserDetailed should be user instead.
activateBonusProgramForPlayer	`activateBonusProgramForPlayer`	No change	
loginUserDetailed		Removed parameter from request: password Added optional parameter in request:	

Methods in old API	Methods in new API	Action taken	Comment
		currencyISOCode	
logoutUser	logoutUser	No change	
getCurrentJackpot	getCurrentJackpot	No change	
depositUserMoney		Removed	Internally calling depositUserMoneyV2
depositUserMoneyV2	depositUserMoneyV3	Removed parameter from request: password	Version upgraded
depositUserMoneyFromChannel		Removed	
depositUserMoneyFromChannelV2		Removed	
depositUserMoneyBypassDepositBonusPrograms		Removed	
depositUserMoneyBypassDepositBonusProgramsV2	depositUserMoneyBypassDepositBonusProgramsV3	Removed parameter from request: password	Version upgraded
withdrawUserMoney		Removed	
withdrawUserMoneyV2	withdrawUserMoneyV3	Removed parameter from request: password	Version upgraded
getUserBonusInfo		Removed	
getUserBonusInfoV2		Removed	
getUserBonusInfoV3	getUserBonusInfoV4	Removed parameter from request: password	Version upgraded
getUserBonusInfoDetail		Removed	
getUserBonusInfoDetailV2		Removed	
getUserBonusInfoDetailV3		Removed	
getUserBonusInfoDetailV4	getUserBonusInfoDetailV5	Removed parameter from request: password	Version upgraded
getBonusTransactionSummary	getBonusTransactionSummary	No change	
giveUserManualBonus		Removed	
giveUserManualBonusV2		Removed	
giveUserManualBonusV4	giveUserManualBonusV5	Removed parameter from request: password	Version upgraded
getIndividualJackpotInfo	getIndividualJackpotInfo	No change	
getUserMoneyBalance		Removed	
getUserMoneyBalanceV2	giveUserManualBonusV3	Removed parameter from request: password	Version upgraded
getTotalBalance	getTotalBalance	No change	
getCoinValues		Removed	

Methods in old API	Methods in new API	Action taken	Comment
getUserClassicTournaments getUserClassicTournamentsV2 getUserClassicTournamentsFrom-Channel getUserClassicTournamentsFrom-ChannelV2 getUserTournaments getUserTournamentsFromChannel getUserTournamentsFromChan-nelV2	getUserTournamentsV2	Added optional parameter in request: channel Removed parameters from response: buyTicketEndD-ate buyTicketStart-Date ticketBased ticketCosts Added parameters to response: channel bonusMoneyEx-piryDate	
getAllClassicTournaments getAllClassicTournamentsV2 getAllClassicTournamentsFromCh-annel getAllClassicTournamentsFromCh-annelV2 getAllTournaments getAllTournamentsFromChannel	getAllTournamentsV2	Added optional parameter in request: channel Removed these parameters from response: buyTicketEndD-ate buyTicketStart-Date ticketBased, ticketCosts Added these parameters to response: channel playerTicketId bonusMoneyEx-piryDate	
isUserInTournament	isUserInTournament	No change	
isUserSessionAlive	isUserSessionAlive	No change	
refreshUserSession	refreshUserSession	No change	
getTournamentDetails		Removed	
getTournamentDetailsV2		Removed	
getTournamentDetailsV3	getTournamentDetailsV3	No change	
getLeaderBoard	getLeaderBoard	No change	
getTournamentResult	getTournamentResult	No change	
purchaseTicket		Removed	
getCurrentOverview	getCurrentOverview	Added an optional para-meter in request, as well as in response: channel	

Methods in old API	Methods in new API	Action taken	Comment
getCurrentOverviewByChannel		Removed this method and merged into getCurrentOverview with optional parameter: channel	
getUserGameStatistics	getUserGameStatistics	Removed this parameter from request: password	
getUserWageredAmount		Removed	
getUserWageredAmountV2	getUserWageredAmountV2	No change	
getUserGameHistory	getUserGameHistory	Removed parameter from request: password	
getUserFrequentPlayerPointsBalance	getUserFrequentPlayerPointsBalance	Removed parameter from request: password	
getUserFrequentPlayerPointsAwarded	getUserFrequentPlayerPointsBalance	Removed parameter from request: password	
updateUserPassword		Removed method	
withdrawUserFrequentPlayerPoints	withdrawUserFrequentPlayerPoints	Removed parameter from request: password	
depositUserFrequentPlayerPoints	depositUserFrequentPlayerPoints	Removed parameter from request: password	
getFrequentPlayerPointsTransactions	getFrequentPlayerPointsTransactions	Removed parameter from request: password	
getAllFrequentPlayerPoints	getAllFrequentPlayerPoints	No change	
executeNamedQueryMatrix	executeNamedQueryMatrix	No change	
executeNamedQuery	executeNamedQuery	No change	
confirmDeposit	confirmDeposit	No change	
confirmWithdrawal	confirmWithdrawal	No change	
getTime	getTime	No change	
getGameIds		Removed	
getGames	getGamesV2	Added additional parameters in response: gameGroupName gameGroupId	
getGameStatistics	getGameStatistics	No change	
getUserFreeRoundGames	getUserFreeRoundGames	Removed para-	

Methods in old API	Methods in new API	Action taken	Comment
		meter from request: password	
getPlayerLimits	getPlayerLimits	Removed parameter from request: password	
setPlayerLimit		Removed	
setPlayerLimit2	setPlayerLimit3	Removed parameter from request: password	Version upgraded
finishUnfinishedGameRoundsForPlayer		Removed	
finishUnfinishedGameRoundsForPlayerV2		Removed	
finishUnfinishedGameRoundsForPlayerV3	finishUnfinishedGameRoundsForPlayerV3	No change	
finishUnfinishedGameRoundsForAllPlayers	finishUnfinishedGameRoundsForAllPlayers	No change	
setPlayerBlockLimitEndDate	setPlayerBlockLimitEndDate	Removed parameter from request: password	
removePlayerLimit	removePlayerLimit	Removed parameter from request: password	
renameUser	renameUser	Removed parameter from request: password	
getAccountingInformation		Removed	
getPlayerCurrencyAccountingInformation		Removed	
convertPlayerToNewCurrency	convertPlayerToNewCurrency	Password parameter removed from request	
setCurrencyExchangeRate	setCurrencyExchangeRate	No change	
getGameInfo	getGameInfo	No change	
hasUserUnfinishedRound	hasUserUnfinishedRound	No change	
getExchangeRates	getExchangeRates	No change	
getGameGroups		Removed	
setPlayersBonusProgramExclusionStatus	setPlayersBonusProgramExclusionStatus	No change	
getDepositBonusAmountForPlayer	getDepositBonusAmountForPlayer	Removed parameter from request: password	
forfeitBonusForPlayer	forfeitBonusForPlayer	Removed parameter from request: password	
getGameWalletInfo	getGameWalletInfo	No change	

Methods in old API	Methods in new API	Action taken	Comment
setPlayerBetLimitForGames	`setPlayerBetLimitForGames`	Removed parameter from request: password	
getPlayerBetLimitForGame	`getPlayerBetLimitForGame`	Removed parameter from request: password	
getPlayerGameBetLimits	`getPlayerGameBetLimits`	Removed parameter from request: password	
removePlayerBetLimitForGame	`removePlayerBetLimitForGame`	Removed parameter from request: password	