Chapter 1

Look Inside

Introduction

© 2018 Gaming Standards Association (GSA)

1.1 Introduction

The Third-Party Game Interface is used to integrate Remote Game Services, Progressive Jackpot Services and iGaming Platforms. The interface includes messages for:

- Discovering and configuring the games available on a Remote Game Service,
- Launching games supported by the Remote Game Service,
- Reporting wagers and wins from the Remote Game Service,
- Communicating state information between Remote Game Content and an iGaming Console,
- Discovering the progressive jackpots available on a Progressive Jackpot Service,
- Reporting contributions to the progressive jackpots on the Progressive Jackpot Service, and
- Managing the award of progressive jackpots by a Remote Game Service.

The Third-Party Game Interface does not include other messages that may be necessary to operate an iGaming website such as messages for logging onto the website, transferring funds to the website, verifying player identification, establishing geo-location, etc. These messages are beyond the scope of this specification.

...(continued)...

© 2018 Gaming Standards Association (GSA)

Chapter 6

Look Inside

Game Play

© 2018 Gaming Standards Association (GSA)

6.1 Introduction

During game play, the player will make wagers and, possibly, receive wins. These activities result in Monetary Transactions between the RGS and the iGP. The Monetary Transactions are generated by the RGS and then authorized or denied by the iGP. The RGC should not display the result of a Monetary Transaction until it has been successfully authorized by the iGP.

The RGS has broad discretion in determining what constitutes a Monetary Transaction. In simple cases, a Monetary Transaction may be a single wager on a spinning reel game or the buy-in to a multi-player poker game. In more complex cases, a Monetary Transaction may encompass a series of wagers made on a sporting event or roulette game. The game design will tend to dictate what constitutes a Monetary Transaction.

The RGS also has broad discretion in determining when to report Monetary Transactions to the iGP. Depending on the game design, wagers and wins may be reported to the iGP at the same time or they may be reported at different times. For example, when spinning reel game is being played, the RGS might report the wager and the win at the same time. In other cases, such as sports betting, the RGS might report the wager at the time that the wager is placed and, subsequently, report the win when the outcome of the sporting event or poker game is known.

When a game allows multiple bets, each bet may be reported to the iGP individually or all bets may be reported as a group. For example, as a player is placing wagers on a roulette game, the RGS might report each wager individually. Alternatively, the RGS might wait until all wagers have been made and then report all wagers at the same time.

6.1.1 Starting and Finishing Game Cycles

The `TPI_startGameCycle` command is used by the RGS to explicitly tell the iGP that a new Game Cycle has started. The `TPI_endGameCycle` command is used by the RGS to explicitly tell the iGP that a Game Cycle has finished. While a Game Cycle is unfinished, the RGS may use the `TPI_moneyTransactions`, `TPI_cancelTransactions`, and `TPI_cancelGameCycle` commands to post and cancel Monetary Transactions associated with the Game Cycle. After a Game Cycle is finished, the `gameCycleFinished` property of the Game Cycle MUST be set to true and subsequent `TPI_moneyTransactions`, `TPI_cancelTransactions`, and `TPI_cancelGameCycle` commands MUST NOT be accepted by the iGP for the Game Cycle.

After the RGS has reported that a Game Cycle has been finished, the RGS MUST NOT attempt to post or cancel any additional Monetary Transactions for the Game Cycle. If the RGS makes such a request, the iGP MUST deny the request and include error code `ERR034 Game Cycle Finished` in its response.

The RGS cannot reopen a Game Cycle by sending a `TPI_startGameCycle` command with the same Game Cycle Identifier. If the RGS makes such a request, the iGP will respond with a logically equivalent `TPI_startGameCycleAck` command, however the Game Cycle will not be reoponed. Once a Game Cycle has been finished and the `gameCycleFinished` property has been set to true, the iGP MUST NOT reopen it.

The `gameCycleExc` property is used to indicate whether the Game Cycle was finished normally — that is, the Game Cycle was played to completion without interruption or errors.

- If the Game Cycle was finished normally, the RGS MUST set the `gameCycleExc` property to `0` (zero).
- If the Game Cycle was not finished normally – for example, due to loss of communications with the RGC – the RGS MUST set the `gameCycleExc` property to a value greater than `0` (zero).
- Once the RGS has reported that the Game Cycle has finished, the RGS cannot change the value of the `gameCycleExc` property. If the RGS makes such a request, the iGP will respond with a logically equivalent `TPI_endGameCycleAck` command, however the Game Cycle Exception will not be changed.

© 2018 Gaming Standards Association (GSA)

Once a Game Cycle has been finished, the iGP MUST NOT change the value of the `gameCycleExc` property.

Once the RGS has reported that a new Game Cycle has started, the RGS MUST always report that the Game Cycle has finished, even if the RGS aborts or otherwise terminates the Game Cycle. Once the RGS has reported that a Game Cycle has started, the RGS MUST report the final outcome of a Game Cycle to the iGP in a `TPI_endGameCycle` command, setting the `gameCycleExc` property to an appropriate value. The RGS must assume that the iGP received the commands used to start the Game Cycle even if the RGS does not receive a response from the iGP.

In addition, if the RGS starts a Game Cycle and then aborts the Game Cycle before any commands associated with the Game Cycle are sent to the iGP, the RGS MUST still report the final outcome of the Game Cycle to the iGP, indicating that the Game Cycle was finished and setting the `gameCycleExc` property to an appropriate value. This requirement assures that the iGP has a complete record of all Game Cycles started by the RGS.

A player may initiate multiple Game Cycles during a Game Session and, thus, may have multiple unfinished Game Cycles at the same time during a Game Session. There is no requirement that one Game Cycle be finished before another Game Cycle begins. In some cases, such as sports betting, all Game Cycles initiated during a Game Session may be unfinished when the Game Session ends.

6.1.2 Retrying Monetary Transactions

Since the RGS must always report the final outcome of a Game Cycle to the iGP, the RGS may have to retry the command used to report the final outcome. It is not expected that the RGS will retry all commands associated with a Game Cycle until acknowledged by the iGP. It is only expected that the RGS will retry the command that reports the final outcome of the Game Cycle until acknowledged — that is, the iGP has responded with the prescribed response or an error code.

For example, the RGS might retry a `TPI_moneyTransactions` command containing the initial wager for a Game Cycle every 2 seconds. If the iGP does not approve or deny the command within 10 seconds, the RGS might abort the Game Cycle, stop retrying the `TPI_moneyTransactions` command, and, instead, start trying a `TPI_endGameCycle` command, indicating that the Game Cycle has finished. To meet the requirement that the final outcome must be reported, the RGS would retry the `TPI_endGameCycle` command until acknowledged by the iGP.

Thus, the iGP MUST be prepared to receive `TPI_endGameCycle` commands that reference Game Cycles that were never reported in previous `TPI_startGameCycle` or `TPI_moneyTransactions` commands. In such cases, the iGP should accept the new Game Cycle as if it had been reported in a previous `TPI_startGameCycle` command; the iGP MUST NOT reject the `TPI_endGameCycle` command simply because it references an unknown Game Cycle.

The iGP MUST also be prepared to receive `TPI_cancelTransactions` commands that contain Monetary Transactions that were not previously approved by the iGP. In such cases, the iGP should authorize and then cancel the Monetary Transactions; the iGP MUST NOT reject the `TPI_cancelTransactions` command simply because it contains a Monetary Transaction that was not previously authorized.

When retrying commands to meet the requirement that the final outcome of all Game Cycles must be reported, the RGS SHOULD send the commands as reconciliation commands. This will help assure that the iGP will still process the commands even though the Game Session might have ended or the Secure Token might have been discarded. See Section 6.1.4 Reconciliation Commands below for more details.

© 2018 Gaming Standards Association (GSA)

6.1.3 Reporting Associated Game Sessions

It is possible for a Game Cycle to be started in one Game Session and finished in another Game Session. For example, the player loses network connectivity and returns to finish a game at a later time. It is also possible for a Game Cycle to be voided or auto-completed outside of a Game Session. For example, the player loses network connectivity and the jurisdictional rules stipulate that the game be voided.

If a new Game Session is started so that a suspended game can be finished by the player, the RGS MUST reference the new Game Session when reporting the Monetary Transactions and other commands required to finish the game, not the Game Session in which the Game Cycle was started.

The new Game Session should have the same properties as the original Game Session, not the properties that would be used if the new Game Session was being started for new Game Cycles. The `brandId`, `skinId`, `gameId`, `betConfigId`, `playerId`, `accountId`, `currencyCode`, `gameType`, `mfgCode`, `themeId`, `paytableId`, `releaseNum`, `affiliateId`, and `jurisdictionCode` properties of the original Game Session MUST be used for the new Game Session. Other properties MAY be different and should reflect the current operating environment in which the Game Session was started.

In addition, when the new Game Session is started, the RGS MUST reference the Game Session in which the Game Cycle was originally started as well as the new Game Session. The original Game Session MUST be reported in the `gameReferenceId` property of the `TPI_startGameSession` command. This creates a cross reference back to the Game Session in which the Game Cycle was originally started. If a new Game Session is being started for new Game Cycles, the `gameReferenceId` property MUST be set to <empty> or omitted from the command.

Once the restored Game Cycle has been finished, the new Game Session MUST be closed. If the player wants to continue playing the same game, another new Game Session MUST be opened for the new Game Cycles. The properties used to start the new Game Sessions for new Game Cycles MUST be based on the properties specified in the Launch URL and `TPI_playerSessionAck` command, not the original Game Session. The new Game Session for new Game Cycles should have properties that reflect the current operating environment.

If the game is auto-completed or voided outside of a Game Session and, thus, a new Game Session is not started, the RGS MUST report the original Game Session — that is, the Game Session in which the Game Cycle was started — when reporting Monetary Transactions and other commands used to finish the Game Cycle. There is no need to start a new Game Session.

6.1.4 Reconciliation Commands

The `TPI_gameCycleExc` property is used to indicate whether a `TPI_moneyTransactions`, `TPI_cancelTransactions`, `TPI_cancelGameCycle, or TPI_endGameCycle` command is being used for reconciliation purposes. When the `gameCycleExc` property is set to a value greater than 0 (zero), the command MAY be used for reconciliation purposes and the `secureToken` property MAY be omitted. When commands are used for reconciliation purposes, it is assumed that the Game Session may have been closed and the Secure Token associated with the Game Session may have been discarded. Therefore, when a command is used for reconciliation purposes, the iGP MUST NOT attempt to validate the Secure Token; the Secure Token MUST be ignored.

6.1.5 Special Transactions

Typically, `TPI_moneyTransactions` commands are used to request debits and credits to a Player Account. However, occasionally, the RGS may need to report information associated with a Game Cycle that does not result in a debit or credit to the Player Account. For example, the RGS may need to report that an in-kind prize has been won, such as a car. In such cases, the RGS will indicate that a Monetary Transaction is a Special Transaction rather than a debit or credit to the Player Account. The iGP MUST handle Special Transactions

© 2018 Gaming Standards Association (GSA)

like Monetary Transactions except that the iGP MUST NOT make a debit or credit to the Player Account. The iGP SHOULD simply include the Special Transactions in its Player Account history and make the information available to the operator as appropriate.

6.1.6 Player Tracking

For player tracking and game performance purposes, the RGS MUST summarize the turnover, theoretical win, actual win, games played, and time played during each Game Cycle and then report that information in the `TPI_endGameCycle` command when the Game Cycle is finished. The methods used to determine this information will vary by Game Type.

© 2018 Gaming Standards Association (GSA)

6.2 Sequence Diagrams

The following sequence diagrams illustrate the sequence of commands used during game play. The first diagram shows a simple scenario, such as spinning reel game, where wagers and wins are reported by the RGS in a single command. The second diagram shows a more complex scenario, such as roulette, where wagers and wins are reported across a series of commands. The third diagram shows an alternate scenario where the initial buy-in for a game, such as multi-player poker, is transferred to the RGS and any remaining funds are transferred back to the iGP when the player leaves the game. Other scenarios are possible for these types of games.

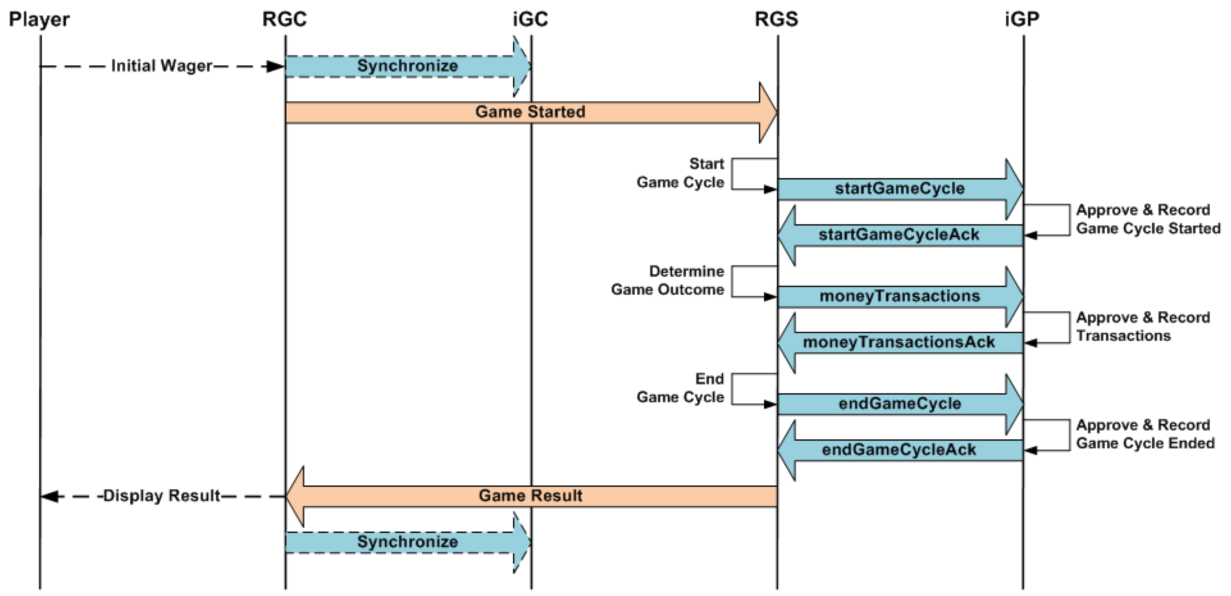
6.2.1 Simple Games

This diagram illustrates the sequence of commands that might be used while a simple game, such as a spinning reel game, is being played.

1. The player selects an amount to wager and then initiates game play.
2. The RGC synchronizes the game state with the iGC, indicating that a game is in progress.
3. The RGC reports to the RGS that a game started.
4. The RGS starts a new Game Cycle, assigning a new unique Game Cycle Identifier, and reports to the iGP that a new game cycle has been started.
5. The iGP acknowledges that the new game cycle has been started.
6. The RGS determines the outcome, creating two Monetary Transactions (one for the wager and one for the win). The RGS assigns a new unique Transaction Identifier to each Monetary Transaction.
7. The RGS sends the two Monetary Transactions to the iGP for approval.
8. The iGP approves the Monetary Transactions and replies to the RGS, including the new Player Account balances in its response.
9. The RGS ends the Game Cycle and reports to the iGP that the game cycle has ended.
10. The iGP acknowledges that the game cycle has ended.
11. The RGS sends the game results to the RGC.
12. The RGC displays the results to the player and then synchronizes the Player Account balances and game state with the iGC, indicating that the game has ended.

© 2018 Gaming Standards Association (GSA)

Figure 6.1 Monetary Transactions for Simple Games



6.2.2 Complex Games

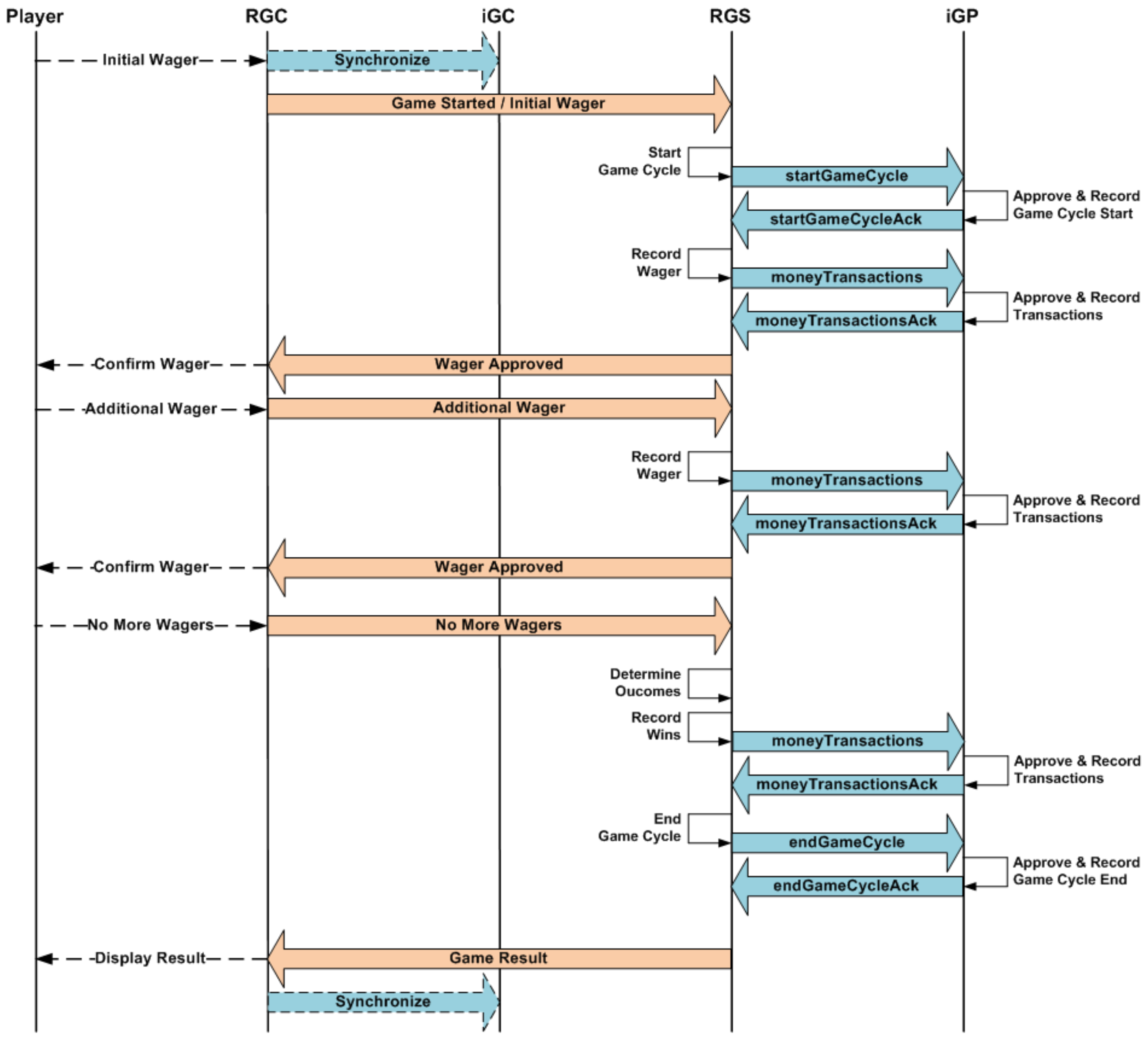
This diagram illustrates the sequence of commands that might be used when a more complex game, such as roulette, is being played. Note that steps 10 through 16 could be repeated multiple times.

1. The player places an initial wager on the game.
2. The RGC synchronizes the game state with the iGC, indicating that a game is in progress.
3. The RGC reports to the RGS that a wager has been placed.
4. The RGS starts a new Game Cycle, assigning a new unique Game Cycle Identifier, and reports to the iGP that a new Game Cycle has started.
5. The iGP acknowledges that the new Game Cycle has started.
6. The RGS creates a Monetary Transaction for the wager, assigning a new unique Transaction Identifier to the Monetary Transaction.
7. The RGS sends the Monetary Transaction to the iGP for approval.
8. The iGP approves the Monetary Transaction and replies to the RGS, including the new Player Account balances in its response.
9. The RGS sends a wager confirmation to the RGC.
10. The RGC confirms the wager to the player.
11. The player places an additional wager on the game.
12. The RGC reports to the RGS that an additional wager has been placed.
13. The RGS creates a Monetary Transaction for the wager, assigning a new unique Transaction Identifier to the Monetary Transaction.
14. The RGS sends the Monetary Transaction to the iGP for approval.
15. The iGP approves the Monetary Transaction and replies to the RGS, including the new Player Account balances in its response.
16. The RGS sends a wager confirmation to the RGC.

© 2018 Gaming Standards Association (GSA)

17. The RGC confirms the wager to the player.
18. The RGS announces to the RGC that no more wagers will be accepted.
19. The RGS determines the outcome of the game and creates a Monetary Transaction for the win, assigning a new unique Transaction Identifier to the Monetary Transaction.
20. The RGS sends the Monetary Transaction to the iGP for approval.
21. The iGP approves the Monetary Transaction and replies to the RGS, including the new Player Account balances in its response.
22. The RGS ends the Game Cycle and reports to the iGP that the Game Cycle has ended.
23. The iGP acknowledges that the Game Cycle has ended.
24. The RGS sends the game outcome to the RGC.
25. The RGC displays the results to the player and then synchronizes the Player Account balances and game state with the iGC, indicating that the game has ended.

Figure 6.2 Monetary Transactions for Complex Games



© 2018 Gaming Standards Association (GSA)

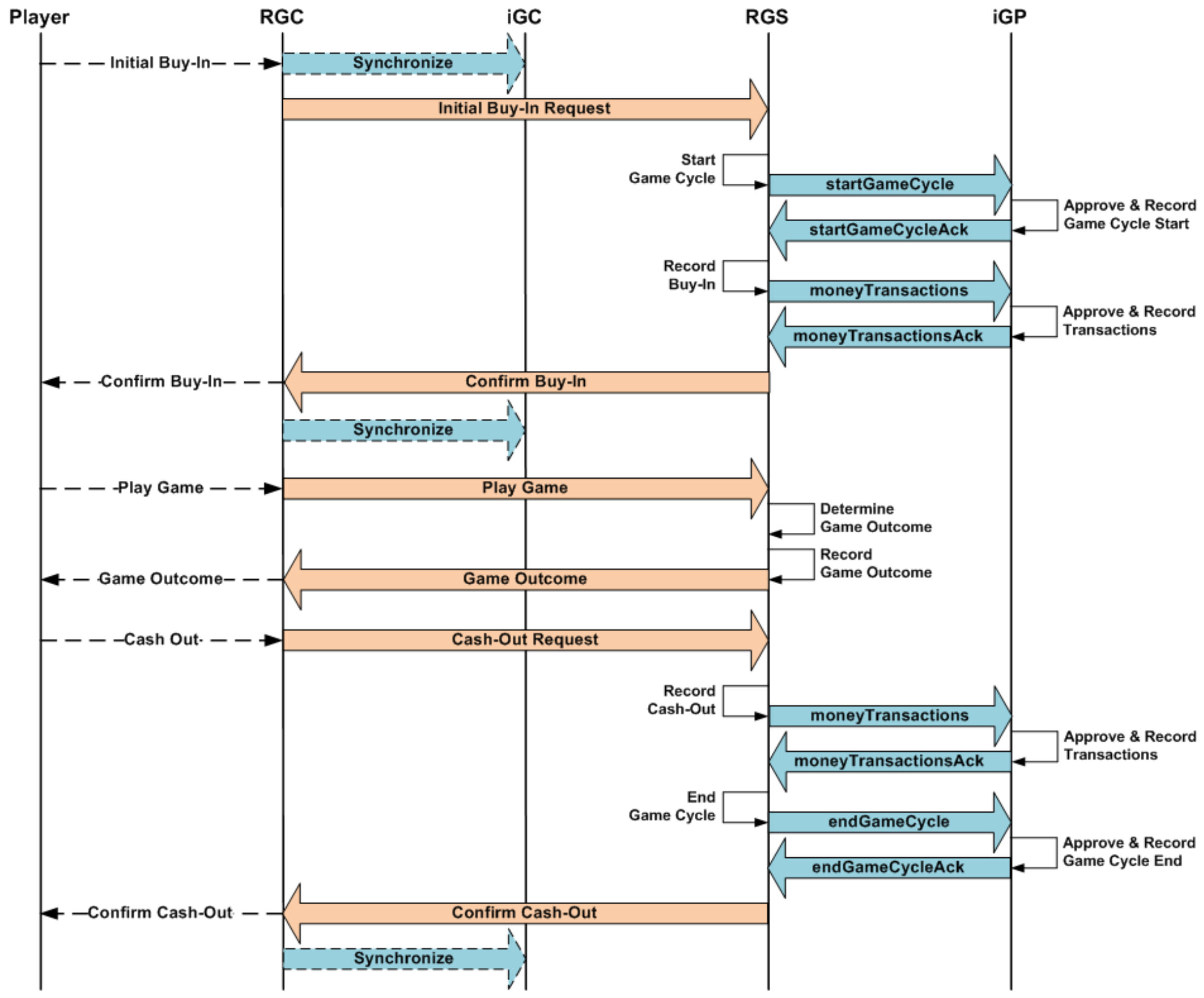
6.2.3 Initial Buy-In Games

This diagram illustrates the sequence of commands that might be used when a game requires an initial buy-in from the player. Any remaining funds are returned to the player's account when the player cashes out and exits the game. Note that some games might allow additional buy-in while the game is being played. In such cases, there would be additional Monetary Transactions between the RGS and the iGP.

1. The player selects an initial buy-in amount and then starts the game.
2. The RGC synchronizes the game state with the iGC, indicating that the game is in progress.
3. The RGC reports the initial buy-in request to the RGS.
4. The RGS starts a new Game Cycle, assigning a new unique Game Cycle Identifier, and reports to the iGP that a new Game Cycle has started.
5. The iGP acknowledges that a new Game Cycle has started.
6. The RGS creates a Monetary Transaction for the buy-in, assigning a new unique Transaction Identifier to the Monetary Transaction.
7. The RGS sends the Monetary Transaction to the iGP for approval.
8. The iGP approves the Monetary Transaction and replies to the RGS, including the new Player Account balances in its response.
9. The RGS sends a buy-in confirmation to the RGC.
10. The RGC confirms the initial buy-in to the player and then synchronizes the Player Account balances with the iGC.
11. The player uses the RGC to play the game offered by the RGS.
12. The player makes a request to cash out.
13. The RGC reports to the RGS that a cash-out has been requested.
14. The RGS creates a Monetary Transaction for the cash-out, assigning a new unique Transaction Identifier to the Monetary Transaction.
15. The RGS sends the Monetary Transaction to the iGP for approval.
16. The iGP approves the Monetary Transaction and replies to the RGS, including the new Player Account balances in its response.
17. The RGS ends the Game Cycle and reports to the iGP that the Game Cycle has ended.
18. The iGP acknowledges that the Game Cycle has ended.
19. The RGS sends a cash-out confirmation to the RGC.
20. The RGC confirms the cash-out to the player and then synchronizes the Player Account balances and game state with the iGC, indicating that the game has ended.

© 2018 Gaming Standards Association (GSA)

Figure 6.3 Monetary Transactions for Initial Buy-In Games



6.2.3.1 Managing Temporary Player Accounts

In this scenario, the RGS will typically create a temporary Player Account to hold the buy-in. The temporary Player Account could be on the RGS or it could be on a dedicated server that is responsible for holding the player's funds. In such cases, the RGS and the dedicated server could use the commands defined within this specification to record the wagers and the wins from the game.

An alternate scenario is also possible. The iGP and iGC could use proprietary commands to set up a temporary Player Account on the iGP to hold the player's funds. In the Launch URL, the iGC would specify the temporary Player Account rather than the primary Player Account. In this case, the RGS would use the commands defined within this specification to record wagers and wins against the temporary Player Account on the iGP. When the player exited the game, the iGP would move any remaining funds from the temporary Player Account back to the primary Player Account.

© 2018 Gaming Standards Association (GSA)

6.2.3.2 Multiple Game Cycles

In this scenario, a single game cycle was used to report both the buy-in and the cash-out for the game. This is an easy way to tie together the buy-in, cash-out, and player tracking information related to the game played on the RGS.

In certain circumstances, however, it might make more sense to report the activity related to the game across multiple game cycles. For example, if the buy-in is held on the RGS for an indefinite period of time and, thus, the player can return to the RGS repeatedly to play the game, it might make sense to report the buy-in in one game cycle, the player tracking information for each visit to the RGS in other game cycles, and the cash-out in a final game cycle.

However, this practice will have an effect on revenue reporting. When devising strategies for reporting the activity related to initial buy-in games, implementers should carefully consider the effect on revenue reporting. (See Chapter 12, Revenue Reporting for details).

© 2018 Gaming Standards Association (GSA)

6.3 TPI_startGameCycle Command

This command is used by the RGS to request that a new Game Cycle be started for a Game Session. A `TPI_startGameCycleAck` command is generated in response to the `TPI_startGameCycle` command. The current value of the Player Account balances are reported by the iGP to the RGS in the `TPI_startGameCycleAck` command.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.1 TPI_startGameCycle Error Codes

Error Code	Description
ERR022	Invalid Secure Token.
ERR023	Incorrect Player Account Identifier for Secure Token.
ERR024	Invalid Game Session Identifier.
ERR028	Transaction Failed (include additional information in `errorMsg`).
ERR031	Incorrect Player Account Identifier for Game Session.
ERR036	Incorrect Player Identifier for Secure Token.
ERR037	Incorrect Player Identifier for Game Session.
ERR041	End Game Session Immediately; Do Not Start New Game Session. [1]
ERR042	End Game Session Immediately; Start New Game Session. [1]

1 See Section 7.4, TPI_closeGameSession Command for more details.

6.3.1 Duplicate Detection

A `TPI_startGameCycle` command is considered to be a duplicate if another `TPI_startGameCycle` command containing the same Game Cycle Identifier (`gameCycleId`) had been previously reported to the iGP and approved by the iGP. If a duplicate `TPI_startGameCycle` command is detected by the iGP, the iGP MUST simply generate a logically equivalent `TPI_startGameCycleAck` response.

Table 6.2 TPI_startGameCycle Properties

Property	Restrictions	Description
secureToken	type: t_secureToken use: required	Secure Token; current value of the Secure Token.
playerId	type: t_playerId use: required	Player Identifier; value of the Player Identifier received in the Game Launch URL.
accountId	type: t_accountId use: required	Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.
gameSessionId	type: t_gameSessionId use: required	Game Session Identifier; identifier assigned by the RGS to the Game Session.

© 2018 Gaming Standards Association (GSA)

Table 6.2 TPI_startGameCycle Properties

Property	Restrictions	Description

`currencyCode`	`type: t_currencyCode`
	`use: required`
		Currency Code; identifies the currency being used for the Game Session.

`brandId`	`type: t_brandId`
	`use: required`
		Brand Identifier; identifies the brand being used for the Game Session.

`skinId`	`type: t_skinId`
	`use: required`
		Skin Identifier; identifies the skin being used for the Game Session.

`localeCode`	`type: t_localeCode`
	`use: required`
		Locale Code; set to the locale (language) currently selected by the player.

`gameCycleId`	`type: t_gameCycleId`
	`use: required`
		Game Cycle Identifier; identifier assigned by the RGS to the game cycle.

`gameGroupId`	`type: t_gameGroupId`
	`use: optional`
	`default: <empty>`
		Game Cycle Group Identifier; identifier assigned by the RGS to a group of game cycles that are dependent of the same decision.

6.3.2 TPI_startGameCycle Example

The following example demonstrates the construction of a `TPI_startGameCycle` command reporting that a new Game Cycle has been started.

```
"command": "TPI_startGameCycle",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameGroupId": ""
}
```

© 2018 Gaming Standards Association (GSA)

6.4 TPI_startGameCycleAck Command

This command is used by the iGP to acknowledge that a new Game Cycle has been approved. The `TPI_startGameCycleAck` command is generated in response to a `TPI_startGameCycle` command.

Table 6.3 TPI_startGameCycleAck Properties

Property	Restrictions	Description

`secureToken`	`type: t_secureToken`
	`use: optional`
		Secure Token; when included, MUST be set to the corresponding value from the request or a new unique Secure Token assigned by the iGP; when omitted, the value of the Secure Token is unchanged.

`playerId`	`type: t_playerId`
	`use: required`
		Player Identifier; MUST be set to the corresponding value from the request.

`accountId`	`type: t_accountId`
	`use: required`
		Player Account Identifier; MUST be set to the corresponding value from the request.

`gameSessionId`	`type: t_gameSessionId`
	`use: required`
		Game Session Identifier; MUST be set to the corresponding value from the request.

`gameCycleId`	`type: t_gameCycleId`
	`use: required`
		Game Cycle Identifier; MUST be set to the corresponding value from the request.

`accountBalance`	`type: accountBalance`
	`use: required`
		Contains account balance information. See Section 5.9, TPI_playerBalance Command for details.

6.4.1 TPI_startGameCycleAck Example

The following example demonstrates the construction of a `TPI_startGameCycleAck` command confirming that a new Game Cycle has been approved.

```
"command": "TPI_startGameCycleAck",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "gameCycleId": "9876FEDC5432BAFE",
    "accountBalance": {
        "playerId": "00101977",
        "accountId": "Z100187",
        "currencyCode": "USD",
        "messageArray": [
            {
                "accountMsg": "Welcome back!"
            },
            {
                "accountMsg": "Good Luck!"
            }
        ],
```

© 2018 Gaming Standards Association (GSA)

```
            "availBalanceAmt": 50000,
            "availFreeSpins": false,
            "balanceArray": [
                {
                    "balanceType": "cashable",
                    "balanceAmt": 50000
                },
                {
                    "balanceType": "cashable",
                    "balanceAmt": 2500,
                    "balanceStatus": "blocked",
                    "balanceRestrict": "Deposit Pending"
                },
                {
                    "balanceType": "nonCashable",
                    "balanceAmt": 10000,
                    "freeSpins": true,
                    "freeSpinId": "be97f52a-9b10-4013-997c-b6b98b6a82d0",
                    "freeSpinCnt": 100,
                    "freeSpinValue": 100
                }
            ]
        }
    }
```

© 2018 Gaming Standards Association (GSA)

6.5 TPI_moneyTransactions Command

This command is used by the RGS to request that a debit or credit be applied to a Player Account. The command may also be used to report Special Transactions. The command does not indicate which Player Account balance should be affected; it only includes the Player Account Identifier. The iGP is responsible for determining which Player Account balance to debit or credit. A `TPI_moneyTransactionsAck` command is generated in response to the `TPI_moneyTransactions` command. The specific Player Account balances that were affected by the `TPI_moneyTransactions` command, as well as the remaining value of the Player Account balanaces, are reported by the iGP in `moneyTransDetail` objects of the `TPI_moneyTransactionsAck` command.

Each `TPI_moneyTransactions` command is idempotent. The iGP MUST approve or deny all Monetary Transactions within the `TPI_moneyTransactions` command as a group; the iGP cannot selectively approve or deny individual Monetary Transactions within a `TPI_moneyTransactions` command.

The RGS SHOULD include a description of the transaction in the `transDesc` property. The description can be very helpful to operators when they are trying to audit game play and resolve disputes with players. The RGS has broad discretion in determining what text to include in the description. However, the RGS SHOULD NOT leave the description empty.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.4 TPI_moneyTransactions Error Codes

Error Code	Description
ERR022	Invalid Secure Token.
ERR023	Incorrect Player Account Identifier for Secure Token.
ERR024	Invalid Game Session Identifier.
ERR025	Insufficient Funds.
ERR026	Player Account Locked.
ERR027	Wager Limit Exceeded.
ERR028	Transaction Failed (include additional information in `errorMsg`).
ERR031	Incorrect Player Account Identifier for Game Session.
ERR032	Incorrect Game Session Identifier for Game Cycle.
ERR034	Game Cycle Finished.
ERR036	Incorrect Player Identifier for Secure Token.
ERR037	Incorrect Player Identifier for Game Session.
ERR041	End Game Session Immediately; Do Not Start New Game Session. [1]
ERR042	End Game Session Immediately; Start New Game Session. [1]

1 See Section 7.4, TPI_closeGameSession Command for more details.

© 2018 Gaming Standards Association (GSA)

After the RGS has reported that a Game Cycle has been finished, the RGS MUST NOT post or cancel any additional Monetary Transactions for the Game Cycle. If the RGS makes such a request, the iGP MUST deny the request and include error code `ERR034 Game Cycle Finished` in its response.

6.5.1 Duplicate Detection

A `moneyTrans` object is considered to be a duplicate if another `moneyTrans` object containing the same Transaction Identifier (`transId`) had been previously approved by the iGP. If a duplicate `moneyTrans` object is detected by the iGP, the iGP MUST NOT debit/credit the Player Account a second time; instead, the iGP MUST simply generate a response that is logically equivalent to the response that was generated when the `moneyTrans` object was first approved; the iGP MUST include the same Reference Identifier (`referenceId`) and Transaction Day (`transDay`) in the response.

Table 6.5 TPI_moneyTransactions Properties

Property	Restrictions	Description

`secureToken`	`type: t_secureToken`
	`use: optional`
	`default: <empty>`
		Secure Token; current value of the Secure Token.

`playerId`	`type: t_playerId`
	`use: required`
		Player Identifier; value of the Player Identifier received in the Game Launch URL.

`accountId`	`type: t_accountId`
	`use: required`
		Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.

`gameSessionId`	`type: t_gameSessionId`
	`use: required`
		Game Session Identifier; identifier assigned by the RGS to the Game Session.

`currencyCode`	`type: t_currencyCode`
	`use: required`
		Currency Code; identifies the currency being used for the Game Session.

`brandId`	`type:t_brandId`
	`use: required`
		Brand Identifier; identifies the brand being used for the Game Session.

`skinId`	`type: t_skinId`
	`use: required`
		Skin Identifier; identifies the skin being used for the Game Session.

`localeCode`	`type: t_localeCode`
	`use: required`
		Locale Code; set to the locale (language) currently selected by the player.

`gameCycleId`	`type: t_gameCycleId`
	`use: required`
		Game Cycle Identifier; identifier assigned by the RGS to the game cycle.

`gameCycleExc`	`type: t_gameCycleCodes`
	`use: optional`
	`default: 0`
		Game Cycle Exception Code.

`gameGroupId`	`type: t_gameGroupId`
	`use: optional`
	`default: <empty>`
		Game Cycle Group Identifier; identifier assigned by the RGS to a group of game cycles that are dependent of the same decision.

Monetary Transactions

© 2018 Gaming Standards Association (GSA)

Table 6.5 TPI_moneyTransactions Properties

Property	Restrictions	Description
`moneyTransArray`	type: array use: required minItems: 0 maxItems: ∞	An arry of `moneyTrans` objects which contain individual debits or credits to the player's account. See Table 6.6, moneyTrans Properties for details.

Table 6.6 moneyTrans Properties

Property	Restrictions	Description
`transSeq`	type: numeric use: required minimum: 1	Transaction Sequence; indicates the sequence in which the transactions should be applied to the player's account; MUST start at 1 and increase by 1 as a strictly increasing series.
`transId`	type: t_transId use: required	Transaction Identifier; assigned by the RGS.
`transAmt`	type: t_money use: required	Transaction Amount; value of the transaction.
`transType`	type: t_transTypes use: required	Transaction Type; indicates whether the transaction is a debit (subtraction) to the player's account, a credit (addition) to the player's account, or a Special Transaction.
`transCategory`	type: t_transCategories use: required	Transaction Category; identifies the general category for the transaction; for example, wager, win, adjustment, etc.
`freeSpinId`	type: t_freeSpinId use: optional default: <empty>	Free Spin Identifier; when a free spin is being reported, identifies the free-spin configuration used for the free spin.
`pjsId`	type: t_pjsId use: optional default: <empty>	Progressive Jackpot Service Identifier; when a progressive win is being reported, identifies the progressive service that made the award.
`controllerId`	type: t_controllerId use: optional default: 0	Jackpot Controller Identifier; when a progressive win is being reported, identifies the jackpot controller associated with the award.
`levelId`	type: t_levelId use: optional default: 0	Jackpot Level Identifier; when a progressive win is being reported, identifies the jackpot level associated with the award.
`pjsDay`	type: t_date use: optional default: <empty>	Jackpot Day; gaming day on which the transaction was approved by the PJS.
`transDesc`	type: t_textMessage use: optional default <empty>	Description of the transaction.

© 2018 Gaming Standards Association (GSA)

6.5.2 TPI_moneyTransactions Examples

The following example demonstrates the construction of a `TPI_moneyTransactions` command reporting that an initial wager has been made.

```
"command": "TPI_moneyTransactions",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameCycleExc": 0,
    "gameGroupId": "",
    "moneyTransArray": [
        {
            "transSeq": 1,
            "transId": "12345601",
            "transAmt": 100,
            "transType": "debit",
            "transCategory": "wager",
            "transDesc": "1 Line 1 Credit"
        }
    ]
}
```

The following example demonstrates the construction of a `TPI_moneyTransactions` command reporting the results of the game cycle.

```
"command": "TPI_moneyTransactions",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameCycleExc": 0,
    "gameGroupId": "",
    "moneyTransArray": [
        {
            "transSeq": 1,
            "transId": "12345602",
            "transAmt": 200,
            "transType": "credit",
            "transCategory": "win",
            "transDesc": "Cherry|Cherry|Cherry"
        }
    ]
}
```

© 2018 Gaming Standards Association (GSA)

The following example demonstrates the construction of a `TPI_moneyTransactions` command reporting the results of the game cycle as a reconciliation command indicating that the game cycle was auto-completed. Note that the Secure Token is not being reported and that the `gameCycleExc` property has been set to 1.

```
"command": "TPI_moneyTransactions",
"data": {
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameCycleExc": 1,
    "gameGroupId": "",
    "moneyTransArray": [
        {
            "transSeq": 1,
            "transId": "12345602",
            "transAmt": 200,
            "transType": "credit",
            "transCategory": "win",
            "transDesc": "Cherry|Cherry|Cherry"
        }
    ]
}
```

© 2018 Gaming Standards Association (GSA)

6.6 TPI_moneyTransactionsAck Command

This command is used by the iGP to approve or deny a set of Monetary Transactions that were received from the RGS. The Monetary Transactions are considered approved if the error code of the response is omitted or set to the <empty> value. The Monetary Transactions are considered denied if the error code of the response is set to a value other than the <empty> value. The `TPI_moneyTransactionsAck` command is generated in response to a `TPI_moneyTransactions` command.

Table 6.7 TPI_moneyTransactionsAck Properties

Property	Restrictions	Description
secureToken	type: t_secureToken use: optional	Secure Token; when included, MUST be set to the corresponding value from the request or a new unique Secure Token assigned by the iGP; when omitted, the value of the Secure Token is unchanged.
playerId	type: t_playerId use: required	Player Identifier; MUST be set to the corresponding value from the request.
accountId	type: t_accountId use: required	Player Account Identifier; MUST be set to the corresponding value from the request.
gameSessionId	type: t_gameSessionId use: required	Game Session Identifier; MUST be set to the corresponding value from the request.
gameCycleId	type: t_gameCycleId use: required	Game Cycle Identifier; MUST be set to the corresponding value from the request.
moneyAckArray	type: array use: required minItems: 0 maxItems: ∞	An array of `moneyAck` objects which contain acknowledgements of individual debits or credits to the player's account. See Table 6.8, moneyAck Properties for details.
accountBalance	type: accountBalance use: required	Contains account balance information. See Section 5.9, TPI_playerBalance Command for details.

Table 6.8 moneyAck Properties

Property	Restrictions	Description
transSeq	type: numeric use: required minimum: 1	Transaction Sequence; MUST be set to the corresponding value from the request.
transId	type: t_transId use: required	Transaction Identifier; MUST be set to the corresponding value from the request.
transAmt	type: t_money use: required	Transaction Amount; MUST be set to the corresponding value from the request.
transType	type: t_transTypes use: required	Transaction Type; MUST be set to the corresponding value from the request.

© 2018 Gaming Standards Association (GSA)

Table 6.8 moneyAck Properties

Property	Restrictions	Description

`transCategory`	`type: t_transCategories`
	`use: required`
		Transaction Category; MUST be set to the corresponding value from the request.

`freeSpinId`	`type: t_freeSpinId`
	`use: optional`
	`default: <empty>`
		Free Spin Identifier; when a free spin is being reported, identifies the free-spin configuration used for the free spin.

`pjsId`	`type: t_pjsId`
	`use: optional`
	`default: <empty>`
		Progressive Jackpot Service Identifier; when a progressive win is being reported, identifies the progressive service that made the award.

`controllerId`	`type: t_controllerId`
	`use: optional`
	`default: 0`
		Jackpot Controller Identifier; when a progressive win is being reported, identifies the jackpot controller associated with the award.

`levelId`	`type: t_levelId`
	`use: optional`
	`default: 0`
		Jackpot Level Identifier; when a progressive win is being reported, identifies the jackpot level associated with the award.

`pjsDay`	`type: t_date`
	`use: optional`
	`default: <empty>`
		Jackpot Day; gaming day on which the transaction was approved by the PJS.

`transDesc`	`type: t_textMessage`
	`use: optional`
	`default <empty>`
		Description of the transaction; MUST be set to the corresponding value from the request.

`referenceId`	`type: t_referenceId`
	`use: required`
		Reference Identifier; value assigned by the iGP to the Monetary Transaction when the Monetary Transaction was approved.

`transDay`	`type: t_date`
	`use: required`
		Transaction Day; gaming day on which the transaction was approved by the iGP.

`moneyDetailArray`	`type: array`
	`use: required`
	`minItems: 0`
	`maxItems: ∞`
		An array of `moneyDetail` objects which identify account balances that were affected by the transaction and the remaining funds after the transaction was completed. See Table 6.9, moneyDetail Properties for details.

Table 6.9 moneyDetail Properties

Property	Restrictions	Description

`balanceType`	`type: t_balanceTypes`
	`use: required`
		Balance identifier; assigned by the iGP; for example, `cashable`.

`balanceSeq`	`type: t_balanceSeq`
	`use: optional`
	`default: 0`
		Balance Sequence; assigned by the iGP; used to identify the sequence of transactions against an account balance.

© 2018 Gaming Standards Association (GSA)

Table 6.9 moneyDetail Properties

Property	Restrictions	Description
detailAmt	type: t_money use: required	Detail Transaction Amount.
detailType	type: t_transTypes use: required	Detail Transaction Type; debit or credit.
balanceAmt	type: t_money use: required	Remaining balance after the monetary transaction.

6.6.1 TPI_moneyTransactionsAck Examples

The following example demonstrates the construction of a `TPI_moneyTransactionsAck` command confirming that an initial wager has been made.

```
"command": "TPI_moneyTransactionsAck",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "gameCycleId": "9876FEDC5432BAFE",
    "moneyAckArray": [
        {
            "transSeq": 1,
            "transId": "12345601",
            "transAmt": 100,
            "transType": "debit",
            "transCategory": "wager",
            "transDesc": "1 Line 1 Credit",
            "referenceId": "98765401",
            "transDay": "2014-10-26",
            "moneyDetailArray": [
                {
                    "balanceType": "cashable",
                    "balanceSeq": 251,
                    "detailAmt": 100,
                    "detailType": "debit",
                    "balanceAmt": 49900
                }
            ]
        }
    ],
    "accountBalance": {
        "playerId": "00101977",
        "accountId": "Z100187",
        "currencyCode": "USD",
        "messageArray": [
            {
                "accountMsg": "Welcome back!"
            },
            {
                "accountMsg": "Good Luck!"
            }
        ],
        "availBalanceAmt": 49900,
        "availFreeSpins": false,
```

© 2018 Gaming Standards Association (GSA)

```
            "balanceArray": [
                {
                    "balanceType": "cashable",
                    "balanceAmt": 49900
                },
                {
                    "balanceType": "cashable",
                    "balanceAmt": 2500,
                    "balanceStatus": "blocked",
                    "balanceRestrict": "Deposit Pending"
                },
                {
                    "balanceType": "nonCashable",
                    "balanceAmt": 10000,
                    "freeSpins": true,
                    "freeSpinId": "be97f52a-9b10-4013-997c-b6b98b6a82d0",
                    "freeSpinCnt": 100,
                    "freeSpinValue": 100
                }
            ]
        }
    }
```

The following example demonstrates the construction of a `TPI_moneyTransactionsAck` command confirming the results of the game cycle.

```
    "command": "TPI_moneyTransactionsAck",
    "data": {
        "secureToken": "A1B2C3D4E5F60718",
        "playerId": "00101977",
        "accountId": "Z100187",
        "gameSessionId": "ABCD1234EFGH5678",
        "gameCycleId": "9876FEDC5432BAFE",
        "moneyAckArray": [
            {
                "transSeq": 1,
                "transId": "12345602",
                "transAmt": 200,
                "transType": "credit",
                "transCategory": "win",
                "transDesc": "Cherry|Cherry|Cherry",
                "referenceId": "98765402",
                "transDay": "2014-10-26",
                "moneyDetailArray": [
                    {
                        "balanceType": "cashable",
                        "balanceSeq": 252,
                        "detailAmt": 200,
                        "detailType": "credit",
                        "balanceAmt": 50100
                    }
                ]
            }
        ],
        "accountBalance": {
            "playerId": "00101977",
            "accountId": "Z100187",
            "currencyCode": "USD",
            "messageArray": [
                {
```

© 2018 Gaming Standards Association (GSA)

```
                    "accountMsg": "Welcome back!"
                },
                {
                    "accountMsg": "Good Luck!"
                }
            ],
            "availBalanceAmt": 50100,
            "availFreeSpins": false,
            "balanceArray": [
                {
                    "balanceType": "cashable",
                    "balanceAmt": 50100
                },
                {
                    "balanceType": "cashable",
                    "balanceAmt": 2500,
                    "balanceStatus": "blocked",
                    "balanceRestrict": "Deposit Pending"
                },
                {
                    "balanceType": "nonCashable",
                    "balanceAmt": 10000,
                    "freeSpins": true,
                    "freeSpinId": "be97f52a-9b10-4013-997c-b6b98b6a82d0",
                    "freeSpinCnt": 100,
                    "freeSpinValue": 100
                }
            ]
        }
    }
```

© 2018 Gaming Standards Association (GSA)

6.7 TPI_cancelTransactions Command

This command is used by the RGS to cancel (void) one or more Monetary Transactions that were previously approved by the iGP. The Monetary Transactions MUST be for the same amount and of the same type (debit or credit) as presented in the original `TPI_moneyTransactions` command. A `TPI_cancelTransactionsAck` command is generated in response to the `TPI_cancelTransactions` command. The specific Player Account balances that were affected by the `TPI_cancelTransactions` command, as well as the remaining value of the Player Account balances, are reported by the iGP in `cancelDetail` objects of the `TPI_cancelTransactionsAck` command.

Each `TPI_cancelTransactions` command is idempotent. The iGP MUST void all Monetary Transactions within the `TPI_cancelTransactions` command as a group; the iGP cannot selectively void individual Monetary Transactions within a `TPI_cancelTransactions` command.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.10 TPI_cancelTransactions Error Codes

Error Code	Description
`ERR022`	Invalid Secure Token.
`ERR023`	Incorrect Player Account Identifier for Secure Token.
`ERR024`	Invalid Game Session Identifier.
`ERR025`	Insufficient Funds.
`ERR026`	Player Account Locked.
`ERR028`	Transaction Failed (include additional information in `errorMsg`).
`ERR029`	Invalid Game Cycle Identifier.
`ERR031`	Incorrect Player Account Identifier for Game Session.
`ERR032`	Incorrect Game Session Identifier for Game Cycle.
`ERR033`	Invalid Transaction Identifier for Game Cycle.
`ERR034`	Game Cycle Finished.
`ERR036`	Incorrect Player Identifier for Secure Token.
`ERR037`	Incorrect Player Identifier for Game Session.
`ERR041`	End Game Session Immediately; Do Not Start New Game Session. [1]
`ERR042`	End Game Session Immediately; Start New Game Session. [1]

1 See Section 7.4, TPI_closeGameSession Command for more details.

After the RGS has reported that a Game Cycle has been finished, the RGS MUST NOT post or cancel any additional Monetary Transactions for the Game Cycle. If the RGS makes such a request, the iGP MUST deny the request and include error code `ERR034 Game Cycle Finished` in its response.

© 2018 Gaming Standards Association (GSA)

6.7.1 Duplicate Detection

A cancelTrans object is considered to be a duplicate if a cancelTrans object containing the same Transaction Identifier (transId) had been previously cancelled by the iGP. If a duplicate cancelTrans object is detected by the iGP, the iGP MUST NOT reverse the Monetary Transaction and update the Player Account a second time; instead, the iGP MUST simply generate a response that is logically equivalent to the response that was generated when the cancelTrans object was first approved; the iGP MUST include the same Cancellation Identifier (cancelId) and Cancellation Day (cancelDay) in the response.

Table 6.11 TPI_cancelTransactions Properties

Property	Restrictions	Description
secureToken	type: t_secureToken use: optional default: <empty>	Secure Token; current value of the Secure Token.
playerId	type: t_playerId use: required	Player Identifier; value of the Player Identifier received in the Game Launch URL.
accountId	type: t_accountId use: required	Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.
gameSessionId	type: t_gameSessionId use: required	Game Session Identifier; identifier assigned by the RGS to the Game Session.
currencyCode	type: t_currencyCode use: required	Currency Code; identifies the currency being used for the Game Session.
brandId	type: t_brandId use: required	Brand Identifier; identifies the brand being used for the Game Session.
skinId	type: t_skinId use: required	Skin Identifier; identifies the skin being used for the Game Session.
localeCode	type: t_localeCode use: required	Locale Code; set to the locale (language) currently selected by the player.
gameCycleId	type: t_gameCycleId use: required	Game Cycle Identifier; identifier assigned by the RGS to the game cycle.
gameCycleExc	type: t_gameCycleCodes use: optional default: 0	Game Cycle Exception Code.
gameGroupId	type: t_gameGroupId use: optional default: <empty>	Game Cycle Group Identifier; identifier assigned by the RGS to a group of game cycles that are dependent of the same decision.
Cancelled Transactions		
cancelTransArray	type: array use: required minItems: 0 maxItems: ∞	An array of cancelTrans objects which contain individual debits or credits to the player's account. See Table 6.12, cancelTrans Properties for details.

© 2018 Gaming Standards Association (GSA)

Table 6.12 cancelTrans Properties

Property	Restrictions	Description
transSeq	type: numeric use: required minimum: 1	Transaction Sequence; indicates the sequence in which the transactions should be applied to the player's account; MUST start at 1 and increase by 1 as a strictly increasing series.
transId	type: t_transId use: required	Transaction Identifier; assigned by the RGS.
transAmt	type: t_money use: required	Transaction Amount; value of the transaction.
transType	type: t_transTypes use: required	Transaction Type; indicates whether the transaction is a debit (subtraction) to the player's account, a credit (addition) to the player's account, or a Special Transaction.
transCategory	type: t_transCategories use: required	Transaction Category; identifies the general category for the transaction; for example, wager, win, adjustment, etc.
freeSpinId	type: t_freeSpinId use: optional default: <empty>	Free Spin Identifier; when a free spin is being reported, identifies the free-spin configuration used for the free spin.
pjsId	type: t_pjsId use: optional default: <empty>	Progressive Jackpot Service Identifier; when a progressive win is being reported, identifies the progressive service that made the award.
controllerId	type: t_controllerId use: optional default: 0	Jackpot Controller Identifier; when a progressive win is being reported, identifies the jackpot controller associated with the award.
levelId	type: t_levelId use: optional default: 0	Jackpot Level Identifier; when a progressive win is being reported, identifies the jackpot level associated with the award.
pjsDay	type: t_date use: optional default: <empty>	Jackpot Day; gaming day on which the transaction was approved by the PJS.
transDesc	type: t_textMessage use: optional default <empty>	Description of the transaction.
referenceId	type: t_referenceId use: required	Reference Identifier; value assigned by the iGP to the Monetary Transaction when the Monetary Transaction was approved.
transDay	type: t_date use: required	Transaction Day; value assigned by the iGP to the Monetary Transaction when the Monetary Transaction was approved.

© 2018 Gaming Standards Association (GSA)

6.7.2 TPI_cancelTransactions Example

The following example demonstrates the construction of a `TPI_cancelTransactions` command requesting that an initial wager be cancelled.

```
"command": "TPI_cancelTransactions",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameCycleExc": 0,
    "gameGroupId": "",
    "cancelTransArray": [
        {
            "transSeq": 1,
            "transId": "12345601",
            "transAmt": 100,
            "transType": "debit",
            "transCategory": "wager",
            "transDesc": "1 Line 1 Credit",
            "referenceId": "98765402",
            "transDay": "2014-10-26"
        }
    ]
}
```

© 2018 Gaming Standards Association (GSA)

6.8 TPI_cancelGameCycle Command

This command is used by the RGS to cancel (void) all Monetary Transactions associated with a specific Game Cycle that were previously approved by the iGP. The RGS does not have to present the previously approved Monetary Transactions in this command. A `TPI_cancelTransactionsAck` command is generated in response to the `TPI_cancelGameCycle` command. The `TPI_cancelTransactionsAck` response MUST include a complete list of all Monetary Transactions that were cancelled at any time for the Game Cycle.

Each `TPI_cancelGameCycle` command is idempotent. The iGP MUST cancel all Monetary Transactions for the Game Cycle as a group or not cancel any Monetary Transactions for the Game Cycle; the iGP cannot selectively cancel individual Monetary Transactions for the Game Cycle.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.13 TPI_cancelGameCycle Error Codes

Error Code	Description
`ERR022`	Invalid Secure Token.
`ERR023`	Incorrect Player Account Identifier for Secure Token.
`ERR024`	Invalid Game Session Identifier.
`ERR025`	Insufficient Funds.
`ERR026`	Player Account Locked.
`ERR028`	Transaction Failed (include additional information in `errorMsg`).
`ERR029`	Invalid Game Cycle Identifier.
`ERR031`	Incorrect Player Account Identifier for Game Session.
`ERR032`	Incorrect Game Session Identifier for Game Cycle.
`ERR034`	Game Cycle Finished.
`ERR036`	Incorrect Player Identifier for Secure Token.
`ERR037`	Incorrect Player Identifier for Game Session.
`ERR041`	End Game Session Immediately; Do Not Start New Game Session. [1]
`ERR042`	End Game Session Immediately; Start New Game Session. [1]

1 See Section 7.4, TPI_closeGameSession Command for more details.

After the RGS has reported that a Game Cycle has been finished, the RGS MUST NOT post or cancel any additional Monetary Transactions for the Game Cycle. If the RGS makes such a request, the iGP MUST deny the request and include error code `ERR034 Game Cycle Finished` in its response.

6.8.1 Duplicate Detection

The Game Cycle specified in the `TPI_cancelGameCycle` command may include one or more Monetary Transactions that were previously cancelled by the iGP. The iGP MUST NOT cancel any such Monetary

© 2018 Gaming Standards Association (GSA)

Transactions a second time. Instead, the iGP MUST simply include logically equivalent responses for the previously cancelled Monetary Transactions; the iGP MUST include the same Cancellation Identifiers (`cancelId`) and Cancellation Days (`cancelDay`) that were reported when the Monetary Transactions were first cancelled.

Table 6.14 TPI_cancelGameCycle Properties

Property	Restrictions	Description

`secureToken`	`type: t_secureToken`
	`use: optional`
	`default: <empty>`
		Secure Token; current value of the Secure Token.

`playerId`	`type: t_playerId`
	`use: required`
		Player Identifier; value of the Player Identifier received in the Game Launch URL.

`accountId`	`type: t_accountId`
	`use: required`
		Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.

`gameSessionId`	`type: t_gameSessionId`
	`use: required`
		Game Session Identifier; identifier assigned by the RGS to the Game Session.

`currencyCode`	`type: t_currencyCode`
	`use: required`
		Currency Code; identifies the currency being used for the Game Session.

`brandId`	`type: t_brandId`
	`use: required`
		Brand Identifier; identifies the brand being used for the Game Session.

`skinId`	`type: t_skinId`
	`use: required`
		Skin Identifier; identifies the skin being used for the Game Session.

`localeCode`	`type: t_localeCode`
	`use: required`
		Locale Code; set to the locale (language) currently selected by the player.

`gameCycleId`	`type: t_gameCycleId`
	`use: required`
		Game Cycle Identifier; identifier assigned by the RGS to the game cycle.

`gameCycleExc`	`type: t_gameCycleCodes`
	`use: optional`
	`default: 0`
		Game Cycle Exception Code.

`gameGroupId`	`type: t_gameGroupId`
	`use: optional`
	`default: <empty>`
		Game Cycle Group Identifier; identifier assigned by the RGS to a group of game cycles that are dependent of the same decision.

6.8.2 TPI_cancelGameCycle Example

The following example demonstrates the construction of a `TPI_cancelGameCycle` command requesting that all Monetary Transactions associated with the game cycle be cancelled.

```
"command": "TPI_cancelGamecycle",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
```

© 2018 Gaming Standards Association (GSA)

```
        "accountId": "Z100187",
        "gameSessionId": "ABCD1234EFGH5678",
        "currencyCode": "USD",
        "brandId": "myBrand",
        "skinId": "mySkin",
        "localeCode": "en-US",
        "gameCycleId": "9876FEDC5432BAFE",
        "gameCycleExc": 0,
        "gameGroupId": ""
    }
```

© 2018 Gaming Standards Association (GSA)

6.9 TPI_cancelTransactionsAck Command

This command is used by the iGP to approve or deny the cancellation of a set of Monetary Transactions that were previously approved by the iGP. The cancellation is considered approved if the error code of the response is omitted or set to the <empty> value. The cancellation is considered denied if the error code of the response is set to a value other than the <empty> value. The `TPI_cancelTransactionsAck` command is generated in response to the `TPI_cancelTransactions` and `TPI_cancelGameCycle` commands.

Table 6.15 TPI_cancelTransactionsAck Properties

Property	Restrictions	Description
`secureToken`	type: `t_secureToken` use: optional	Secure Token; when included, MUST be set to the corresponding value from the request or a new unique Secure Token assigned by the iGP; when omitted, the value of the Secure Token is unchanged.
`playerId`	type: `t_playerId` use: required	Player Identifier; MUST be set to the corresponding value from the request.
`accountId`	type: `t_accountId` use: required	Player Account Identifier; MUST be set to the corresponding value from the request.
`gameSessionId`	type: `t_gameSessionId` use: required	Game Session Identifier; MUST be set to the corresponding value from the request.
`gameCycleId`	type: `t_gameCycleId` use: required	Game Cycle Identifier; MUST be set to the corresponding value from the request.
`cancelAckArray`	type: `array` use: required minItems: 0 maxItems: ∞	An array of `cancelAck` objects which contain acknowledgements of individual debits or credits to the player's account. See Table 6.16, cancelAck Properties for details.
`accountBalance`	type: accountBalance use: required	Contains account balance information. See Section 5.9, TPI_playerBalance Command for details.

Table 6.16 cancelAck Properties

Property	Restrictions	Description
`transSeq`	type: `numeric` use: required minimum: 1	Transaction Sequence; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, indicates the sequence in which the transactions were applied to the player's account; MUST start at 1 and increase by 1 as a strictly increasing series.
`transId`	type: `t_transId` use: required	Transaction Identifier; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

© 2018 Gaming Standards Association (GSA)

Table 6.16 cancelAck Properties

Property	Restrictions	Description

`transAmt`	`type: t_money`
	`use: required`
		Transaction Amount; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`transType`	`type: t_transTypes`
	`use: required`
		Transaction Type; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`transCategory`	`type: t_transCategories`
	`use: required`
		Transaction Category; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`freeSpinId`	`type: t_freeSpinId`
	`use: optional`
	`default: <empty>`
		Free Spin Identifier; when a free spin is being reported, identifies the free-spin configuration used for the free spin.

`pjsId`	`type: t_pjsId`
	`use: optional`
	`default: <empty>`
		Progressive Jackpot Service; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`controllerId`	`type: t_controllerId`
	`use: optional`
	`default: 0`
		Jackpot Controller Identifier; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`levelId`	`type: t_levelId`
	`use: optional`
	`default: 0`
		Jackpot Level Identifier; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`pjsDay`	`type: t_date`
	`use: optional`
	`default: <empty>`
		Jackpot Day; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

`transDesc`	`type: t_textMessage`
	`use: optional`
	`default <empty>`
		Description of the transaction; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value from the Monetary Transaction.

© 2018 Gaming Standards Association (GSA)

Table 6.16 cancelAck Properties

Property	Restrictions	Description

`referenceId`	`type: t_referenceId`
	`use: required`
		Reference Identifier; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value assigned by the iGP to the Monetary Transaction.

`transDay`	`type: t_date`
	`use: required`
		Transaction Day; in the case of `cancelTransactions`, MUST be set to the corresponding value from the request; in the case of `cancelGameCycle`, MUST be set to the value assigned by the iGP to the Monetary Transaction.

`cancelId`	`type: t_referenceId`
	`use: required`
		Cancellation Identifier; value assigned by the iGP to the Monetary Transaction when the Monetary Transaction was cancelled.

`cancelDay`	`type: t_date`
	`use: required`
		Cancellation Day; gaming day on which the Monetary Transaction was cancelled by the iGP.

`cancelDetailArray`	`type: array`
	`use: required`
	`minItems: 0`
	`maxItems: ∞`
		An array of `cancelDetail` objects which identify account balances that were affected by the cancellation and the remaining value after the cancellation was completed. See Table 6.17, cancelDetail Properties for details.

Table 6.17 cancelDetail Properties

Property	Restrictions	Description

`balanceType`	`type: t_balanceTypes`
	`use: required`
		Balance identifier; assigned by the iGP; for example, `cashable`.

`balanceSeq`	`type: t_balanceSeq`
	`use: optional`
	`default: 0`
		Balance Sequence; assigned by the iGP; used to identify the sequence of transactions against an account balance.

`detailAmt`	`type: t_money`
	`use: required`
		Detail Transaction Amount.

`detailType`	`type: t_transTypes`
	`use: required`
		Detail Transaction Type; debit or credit.

`balanceAmt`	`type: t_money`
	`use: required`
		Remaining balance after the cancellation of the monetary transaction.

6.9.1 TPI_cancelTransactionsAck Example

The following example demonstrates the construction of a `TPI_cancelTransactionsAck` command confirming that an initial wager has been canceled.

```
   "command": "TPI_cancelTransactionsAck",
```

© 2018 Gaming Standards Association (GSA)

```
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "gameCycleId": "9876FEDC5432BAFE",
    "cancelAckArray": [
        {
            "transSeq": 1,
            "transId": "12345601",
            "transAmt": 100,
            "transType": "debit",
            "transCategory": "wager",
            "transDesc": "1 Line 1 Credit",
            "referenceId": "98765401",
            "transDay": "2014-10-26",
            "cancelId": "98765402",
            "cancelDay": "2014-10-26",
            "cancelDetailArray": [
                {
                    "balanceType": "cashable",
                    "balanceSeq": 252,
                    "detailAmt": 100,
                    "detailType": "credit",
                    "balanceAmt": 50000
                }
            ]
        }
    ],
    "accountBalance": {
        "playerId": "00101977",
        "accountId": "Z100187",
        "currencyCode": "USD",
        "messageArray": [
            {
                "accountMsg": "Welcome back!"
            },
            {
                "accountMsg": "Good Luck!"
            }
        ],
        "availBalanceAmt": 50000,
        "availFreeSpins": false,
        "balanceArray": [
            {
                "balanceType": "cashable",
                "balanceAmt": 50000
            },
            {
                "balanceType": "cashable",
                "balanceAmt": 2500,
                "balanceStatus": "blocked",
                "balanceRestrict": "Deposit Pending"
            },
            {
                "balanceType": "nonCashable",
                "balanceAmt": 10000,
                "balanceStatus": "blocked",
                "balanceRestrict": "Required Wagers Not Made",
                "freeSpins": true,
                "freeSpinId": "be97f52a-9b10-4013-997c-b6b98b6a82d0",
```

© 2018 Gaming Standards Association (GSA)

```
                    "freeSpinCnt": 100,
                    "freeSpinValue": 100
                }
            ]
        }
    }
```

© 2018 Gaming Standards Association (GSA)

6.10 TPI_endGameCycle Command

This command is used by the RGS to report that a Game Cycle has ended. A `TPI_endGameCycleAck` command is generated in response to the `TPI_endGameCycle` command.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.18 TPI_endGameCycle Error Codes

Error Code	Description
`ERR022`	Invalid Secure Token.
`ERR023`	Incorrect Player Account Identifier for Secure Token.
`ERR024`	Invalid Game Session Identifier.
`ERR028`	Transaction Failed (include additional information in `errorMsg`).
`ERR029`	Invalid Game Cycle Identifier.
`ERR031`	Incorrect Player Account Identifier for Game Session.
`ERR032`	Incorrect Game Session Identifier for Game Cycle.
`ERR036`	Incorrect Player Identifier for Secure Token.
`ERR037`	Incorrect Player Identifier for Game Session.

6.10.1 Duplicate Detection

If the Game Cycle specified in the `TPI_endGameCycle` command is already finished, the iGP MUST simply respond with a logically equivalent `TPI_endGameCycleAck` response; the iGP MUST include the same Revenue Day (`revenueDay`) and Game Cycle Exception (`gameCycleExc`) that were reported when the Monetary Transactions were first ended.

Table 6.19 TPI_endGameCycle Properties

Property	Restrictions	Description

`secureToken`	`type: t_secureToken`
	`use: optional`
	`default: <empty>`
		Secure Token; current value of the Secure Token.

`playerId`	`type: t_playerId`
	`use: required`
		Player Identifier; value of the Player Identifier received in the Game Launch URL.

`accountId`	`type: t_accountId`
	`use: required`
		Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.

`gameSessionId`	`type: t_gameSessionId`
	`use: required`
		Game Session Identifier; identifier assigned by the RGS to the Game Session.

© 2018 Gaming Standards Association (GSA)

Table 6.19 TPI_endGameCycle Properties

Property	Restrictions	Description

`currencyCode`	`type: t_currencyCode`
	`use: required`
		Currency Code; identifies the currency being used for the Game Session.

`brandId`	`type: t_brandId`
	`use: required`
		Brand Identifier; identifies the brand being used for the Game Session.

`skinId`	`type: t_skinId`
	`use: required`
		Skin Identifier; identifies the skin being used for the Game Session.

`localeCode`	`type: t_localeCode`
	`use: required`
		Locale Code; set to the locale (language) currently selected by the player.

`gameCycleId`	`type: t_gameCycleId`
	`use: required`
		Game Cycle Identifier; identifier assigned by the RGS to the game cycle.

`gameCycleExc`	`type: t_gameCycleCodes`
	`use: optional`
	`default: 0`
		Game Cycle Exception Code.

`gameGroupId`	`type: t_gameGroupId`
	`use: optional`
	`default: <empty>`
		Game Cycle Group Identifier; identifier assigned by the RGS to a group of game cycles that are dependent of the same decision.

Player Tracking Information		

`turnoverAmt`	`type: t_money`
	`use: required`
		Turnover Amount; the total of all player wagers for the Game Cycle; set to 0 if the Game Cycle was cancelled.

`theoreticalWin`	`type: t_money`
	`use: required`
		Theoretical Win; the total player theoretical win from all player wagers for the Game Cycle; set to 0 if Game Cycle was cancelled.

`baseWin`	`type: t_money`
	`use: required`
		Base Win; the total of all base paytable wins for the Game Cycle (`win` or `inKindWin`); set to 0 if Game Cycle was cancelled.

`progWin`	`type: t_money`
	`use: optional`
	`default: 0`
		Standard Jackpot Win; the total of all standard progressive jackpot wins for the Game Cycle (`progWin` or `progInKindWin`); set to 0 if Game Cycle was cancelled.

`mysteryWin`	`type: t_money`
	`use: optional`
	`default: 0`
		Mystery Jackpot Win; the total of all mystery progressive jackpot wins for the Game Cycle (`mysteryWin` or `mysteryInKindWin`); set to 0 if Game Cycle was cancelled.

`bonusWin`	`type: t_money`
	`use: optional`
	`default: 0`
		Bonus Win; the total of all bonus wins for the Game Cycle (`bonusWin` or `bonusInKindWin`); set to 0 if Game Cycle was cancelled.

`timePlayed`	`type: t_quantity`
	`use: required`
		Time Played; the total time played (in seconds) during the Game Cycle; set to 0 if Game Cycle was cancelled.

© 2018 Gaming Standards Association (GSA)

Table 6.19 TPI_endGameCycle Properties

Property	Restrictions	Description
gamesPlayed	type: t_quantity use: required	Game Played; the total number of games played by the player during the Game Cycle; set to 0 if Game Cycle was cancelled.

6.10.2 TPI_endGameCycle Example

The following example demonstrates the construction of an `TPI_endGameCycle` command reporting that a Game Cycle has ended.

```
"command": "TPI_endGameCycle",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "currencyCode": "USD",
    "brandId": "myBrand",
    "skinId": "mySkin",
    "localeCode": "en-US",
    "gameCycleId": "9876FEDC5432BAFE",
    "gameCycleExc": 0,
    "gameGroupId": "",
    "turnoverAmt": 100,
    "theoreticalWin": 96,
    "baseWin": 200,
    "timePlayed": 4,
    "gamesPlayed": 1
}
```

© 2018 Gaming Standards Association (GSA)

6.11 TPI_endGameCycleAck Command

This command is used by the iGP to acknowledge that a Game Cycle has been finished. The `TPI_endGameCycleAck` command is generated in response to the `TPI_endGameCycle` command.

Table 6.20 TPI_endGameCycleAck Properties

Property	Restrictions	Description
`secureToken`	type: `t_secureToken` use: optional	Secure Token; when included, MUST be set to the corresponding value from the request or a new unique Secure Token assigned by the iGP; when omitted, the value of the Secure Token is unchanged.
`playerId`	type: `t_playerId` use: required	Player Identifier; MUST be set to the corresponding value from the request.
`accountId`	type: `t_accountId` use: required	Player Account Identifier; MUST be set to the corresponding value from the request.
`gameSessionId`	type: `t_gameSessionId` use: required	Game Session Identifier; MUST be set to the corresponding value from the request.
`gameCycleId`	type: `t_gameCycleId` use: required	Game Cycle Identifier; MUST be set to the corresponding value from the request.
`revenueDay`	type: `t_date` use: required	Revenue Day; gaming day on which the game cycle was finished; assigned by the iGP when the game is finished.
`accountBalance`	type: accountBalance use: required	Contains account balance information. See Section 5.9, TPI_playerBalance Command for details.

6.11.1 TPI_endGameCycleAck Example

The following example demonstrates the construction of a `TPI_endGameCycleAck` command confirming that a Game Cycle has been finished.

```
"command": "TPI_endGameCycleAck",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678",
    "gameCycleId": "9876FEDC5432BAFE",
    "revenueDay": "2016-03-15",
    "accountBalance": {
        "playerId": "00101977",
        "accountId": "Z100187",
        "currencyCode": "USD",
        "messageArray": [
            {
                "accountMsg": "Welcome back!"
```

© 2018 Gaming Standards Association (GSA)

```
            },
            {
                "accountMsg": "Good Luck!"
            }
        ],
        "availBalanceAmt": 50000,
        "availFreeSpins": false,
        "balanceArray": [
            {
                "balanceType": "cashable",
                "balanceAmt": 50000
            },
            {
                "balanceType": "cashable",
                "balanceAmt": 2500,
                "balanceStatus": "blocked",
                "balanceRestrict": "Deposit Pending"
            },
            {
                "balanceType": "nonCashable",
                "balanceAmt": 10000,
                "balanceStatus": "blocked",
                "balanceRestrict": "Required Wagers Not Made",
                "freeSpins": true,
                "freeSpinId": "be97f52a-9b10-4013-997c-b6b98b6a82d0",
                "freeSpinCnt": 100,
                "freeSpinValue": 100
            }
        ]
    }
}
```

© 2018 Gaming Standards Association (GSA)

6.12 TPI_keepAlive Command

This command is used by the RGS to prevent the iGP from terminating a Game Session due to inactivity. This command may be the only way that the iGP knows that the game is still in progress. Operational and jurisdictional requirements will dictate how frequently this command should be generated. A `TPI_keepAliveAck` command is generated in response to the `TPI_keepAlive` command.

In addition to other error codes that the iGP may report, the iGP may report the following error codes, as appropriate, indicating that the requested action was not taken.

Table 6.21 TPI_keepAlive Error Codes

Error Code	Description
`ERR022`	Invalid Secure Token.
`ERR023`	Incorrect Player Account Identifier for Secure Token.
`ERR024`	Invalid Game Session Identifier.
`ERR031`	Incorrect Player Account Identifier for Game Session.
`ERR036`	Incorrect Player Identifier for Secure Token.
`ERR037`	Incorrect Player Identifier for Game Session.
`ERR041`	End Game Session Immediately; Do Not Start New Game Session. [1]
`ERR042`	End Game Session Immediately; Start New Game Session. [1]

1 See Section 7.4, TPI_closeGameSession Command for more details.

Table 6.22 TPI_keepAlive Properties

Property	Restrictions	Description

`secureToken`	`type: t_secureToken`
	`use: required`
		Secure Token; current value of the Secure Token.

`playerId`	`type: t_playerId`
	`use: required`
		Player Identifier; value of the Player Identifier received in the Game Launch URL.

`accountId`	`type: t_accountId`
	`use: required`
		Player Account Identifier; value of the Player Account Identifier received in the Game Launch URL.

`gameSessionId`	`type: t_gameSessionId`
	`use: required`
		Game Session Identifier; identifier assigned by the RGS to the Game Session.

6.12.1 TPI_keepAlive Example

The following example demonstrates the construction of a `TPI_keepAlive` command.

```
"command": "TPI_keepAlive",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
```

© 2018 Gaming Standards Association (GSA)

```
        "playerId": "00101977",
        "accountId": "Z100187",
        "gameSessionId": "ABCD1234EFGH5678"
    }
```

© 2018 Gaming Standards Association (GSA)

6.13 TPI_keepAliveAck Command

This command is used by the iGP to acknowledge receipt of a keep-alive request from the RGS. The `TPI_keepAliveAck` command is generated in response to a `TPI_keepAlive` command.

Table 6.23 TPI_keepAliveAck Properties

Property	Restrictions	Description
`secureToken`	type: `t_secureToken` use: optional	Secure Token; when included, MUST be set to the corresponding value from the request or a new unique Secure Token assigned by the iGP; when omitted, the value of the Secure Token is unchanged.
`playerId`	type: `t_playerId` use: required	Player Identifier; MUST be set to the corresponding value from the request.
`accountId`	type: `t_accountId` use: required	Player Account Identifier; MUST be set to the corresponding value from the request.
`gameSessionId`	type: `t_gameSessionId` use: required	Game Session Identifier; MUST be set to the corresponding value from the request.

6.13.1 keepAliveAck Example

The following example demonstrates the construction of a `keepAliveAck` command.

```
"command": "keepAliveAck",
"data": {
    "secureToken": "A1B2C3D4E5F60718",
    "playerId": "00101977",
    "accountId": "Z100187",
    "gameSessionId": "ABCD1234EFGH5678"
}
```

© 2018 Gaming Standards Association (GSA)

© 2018 Gaming Standards Association (GSA)