Radical Blue Gaming
Incredibly Innovative Gaming Solutions

Release Notes – RadBlue System Tester (RST)

Version 1.8 [released: December 8, 2008]

High-Level Summary

In this release, we added support for installable packages as well as an ID database for the WAT Transfer Insert ID function. In addition, RST now accepts GZIP messages.

New Features

- RST now supports installable package files. For more information on using installable packages as well as creating your own installable packages for use with the RST SmartEGM, see Bulletin 03: Installable Packages.
    - The SmartEGM can download and install packages containing chunks of SmartEGM configuration files expand to become actual gamePlay devices in the SmartEGM. The process by which packages are installed is:
        1. A parser analyzes the contents of the package and then verifies that the package is semantically valid and complete with respect to the **smartegm-config.xml** schema.
        2. If this package is installable, a `commsClosing` command is sent to all active hosts, and then the new gamePlay devices are installed, updating the SmartEGM data model. The new devices are appended to the end of list of game play devices (starting at the next available *deviceId*), and are all owned by the EGM.
        3. RST is then restarted. On restart, the data model and all references are verified. A `commsOnline` command is then sent to all registered hosts.
        4. The `commConfig` host can then use `commConfig` commands to set the owner, and the configurator host values for these new devices, after which a `commsOnline` command is automatically sent to these newly assigned hosts.
        5. The configuration host can then use the appropriate `optionConfig` commands to configure these new devices.
    - In the **package.xml** file, module-type equals a "G2S_game" indicates there are new gamePlay devices in the modules file.
    - The *filename* attribute must point to a SmartEGM configuration file in the module directory portion of the package.

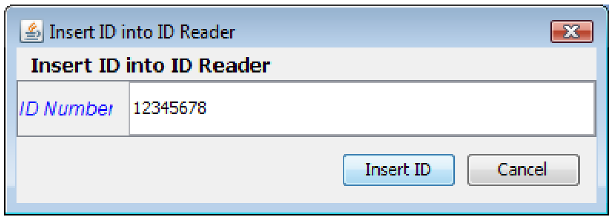
- A sample RadBlue installable package is available for download from the RadBlue web site: http://www.radblue.com/downloads/smartegm/packages/package-4.zip
- The package payload is a **smartegm-config.xml** file, containing one or more gamePlay devices. The device IDs used in the file will be replaced when the devices are added to the SmartEGM data model, but are still required to be non-zero and unique within this file.
- When a RadBlue package is added to the EGM, if the **package.xml** file indicates the modules it contains can be installed, the package validation ensures that only gamePlay devices are contained in the payload and that the device IDs are non-zero and unique. The **smartegm-config.xml** snippet is then validated against the schema for SmartEGM configuration files.
- New gamePlay devices installed by this process are considered SmartEGM devices, so they appear in the descriptor list and are available for configuration.

Improvements

- Support for an ID database has been added to the Insert ID option on the SmartEGM WAT Transfers tab. This is the same database used to populate the Insert ID option on the Player Verbs tab.

  The **id-rst.xml** database is created when you install RST, and automatically provides a randomly selected ID number when you click **Insert ID** under the SmartEGM WAT Transfers tab. By default, this database is populated with the following IDs:

  - 22222222
  - 12345678
  - 11111111



ID Number: 12345678

  The **id-rst.xml** database is used with the SmartEGM user interface only. It is separate from the databases used with Tiger verbs.

- RST now accepts GZIP messages, which are compressed into HTTP_STACK, to the same path as non-GZIP messages.
- The `getTransportOptions` command has been modified to return NO_GZIP and GZIP_IN_HTTP_STACK.

Corrections

- Errors for the `eventHandler` command were being sent with a *commandId* of zero (0). Now, errors are correctly assigned the next *commandId* for the applicable host.

- The S2S simulator now correctly reverses the *fromSystem* and *toSystem* attribute values in the `s2sAck` command.

- The SmartEGM now correctly handles unsupported events.

Version 1.7 [released: November 3, 2008]

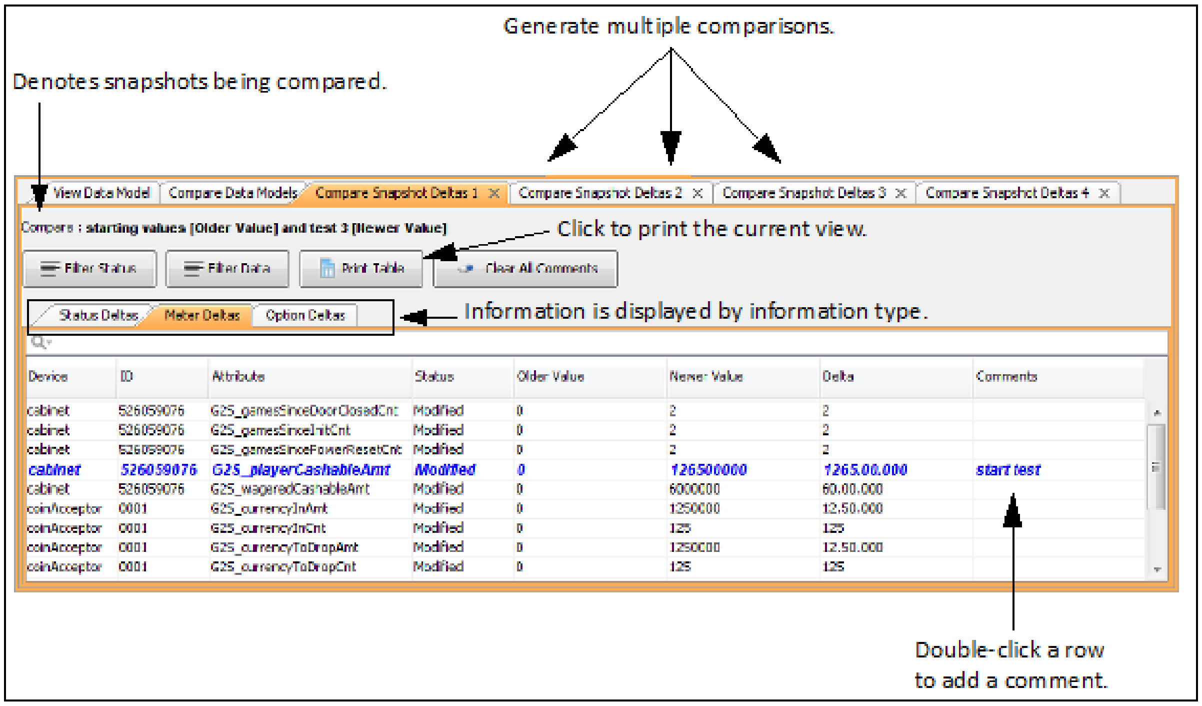
High-Level Summary

In release 1.7, we have added a new Snapshot and Compare feature that allows you to capture EGM data model meters, options, and status values at different points in your testing, and compare them against one another. In addition, we've added Tiger script support for an ID database and a voucher database.

New Features

- A Snapshot and Compare feature has been added to the RST SmartEGM.

  To use the Snapshot and Compare feature, go to: **SmartEGM** > **Data Model Viewer** > **View Data Model**, and click **Snapshot**. When you have at least two snapshots, click the **Compare Data Models** tab, select two snapshots, and click **Compare**. The **Compare Snapshot Deltas** screen displays the comparison (below).


Generate multiple comparisons.
Denotes snapshots being compared.
View Data Model
Compare Data Models
Compare Snapshot Deltas 1
Compare Snapshot Deltas 2
Compare Snapshot Deltas 3
Compare Snapshot Deltas 4
Click to print the current view.
Filter Status
Filter Data
Print Table
Clear All Comments
Status Deltas
Meter Deltas
Option Deltas
Information is displayed by information type.
Double-click a row to add a comment.

Compare : starting values [Older Value] and test 3 [Newer Value]

Device	ID	Attribute	Status	Older Value	Newer Value	Delta	Comments
cabinet	526059076	G2S_gamesSinceDoorClosedCnt	Modified	0	2	2	
cabinet	526059076	G2S_gamesSinceInitCnt	Modified	0	2	2	
cabinet	526059076	G2S_gamesSincePowerResetCnt	Modified	0	2	2	
cabinet	*526059076*	*G2S_playerCashableAmt*	*Modified*	*0*	*126500000*	*1265.00.000*	*start test*
cabinet	526059076	G2S_wageredCashableAmt	Modified	0	6000000	60.00.000	
coinAcceptor	0001	G2S_currencyInAmt	Modified	0	1250000	12.50.000	
coinAcceptor	0001	G2S_currencyInCnt	Modified	0	125	125	
coinAcceptor	0001	G2S_currencyToDropAmt	Modified	0	1250000	12.50.000	
coinAcceptor	0001	G2S_currencyToDropCnt	Modified	0	125	125	

  For more information, see the *Snapshot and Compare* document.

- A new Tiger verb, `tiger:DataModel.snapshot`, has been added. This verb allows you to take a snapshot of the EGM data model at any time during a Tiger script. See About the tiger:DataModel.snapshot Verb for more information.

Improvements

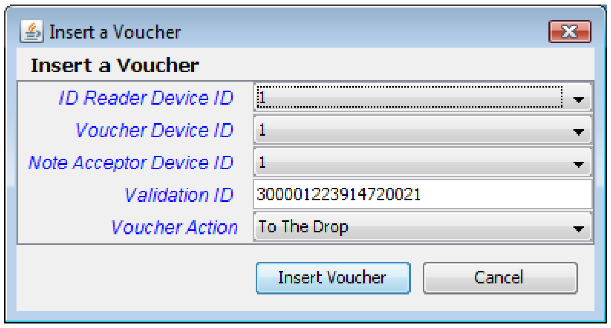
- The following Tiger verbs have been added to RST:
    - `Human.insertIDFromDatabase`
    - `Human.removeIDToDatabase`
    - `Human.insertVoucherFromDatabase`
    - `Human.createVoucherToDatabase`
    - `if-voucher-available`

  For information on using the new tiger verbs, see Using Tiger Scripts with a Voucher Database and Using Tiger Scripts with an ID Database.

  For detailed information on each verb, see ID Database and Voucher Database Tiger Verbs.

  For information on the example Tiger scripts installed with the SmartEGM, see About the Example Tiger Scripts.

- Support for a voucher database has been added to the SmartEGM Player Verbs user interface.



Insert a Voucher

ID Reader Device ID	1
Voucher Device ID	1
Note Acceptor Device ID	1
Validation ID	300001223914720021
Voucher Action	To The Drop

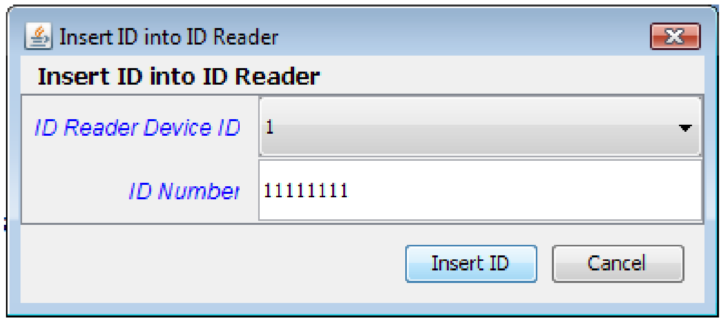
The voucher database is a file of voucher numbers that are used to validate redeemed vouchers.

The **voucher-rst.xml** is created automatically when you click **Create Voucher** under the SmartEGM Player Verbs tab. Each time you click **Create Voucher**, another voucher is inserted into the database

When you click **Insert Voucher**, a randomly selected voucher is removed from the voucher database and is used as the default validation ID.

The **voucher-rst.xml** database is used with the SmartEGM user interface. It is separate from the databases used with the new Tiger verbs.

For more details on the location of the databases and how to use them with your host system's data, see Using Tiger Scripts with an ID Database and Using Tiger Scripts with a Voucher Database.



Insert ID into ID Reader

ID Reader Device ID	1
ID Number	11111111

- Support for an ID database has been added to SmartEGM Player Verbs user interface.

The ID database is a file of player IDs.

The **id-rst.xml** database is created when you install RST, and automatically provides a randomly selected ID number when you click **Insert ID** under the SmartEGM Player Verbs tab.

By default, this database is populated with the following player IDs:

- o 22222222
- o 12345678
- o 11111111

The **id-rst.xml** database is used with the SmartEGM user interface only. It is separate from the databases used with the new Tiger verbs.

- In accordance with the G2S Specification, the SmartEGM no longer allows a device ID equal to zero (0). Therefore, `logList` and `logStatus` requests and responses must be constructed using a non-zero device ID that the requestor can access. This change affects the log commands in all classes.

  Example 1

  Previously, if a host requested the gamePlay recall log through *deviceId*=4, the SmartEGM would return the log using *deviceId*=0 to indicate the log is for the class, rather than just one device. Now, the *deviceId* of the request is used in the response even though the response represents class-level information.

  Example 2

  Previously, SmartEGM allowed a host to request the `logStatus` or `logList` using *deviceId*=0. Now, the *deviceId* must be greater than zero (0). This also allows SmartEGM to determine whether the requesting host has guest access.

- The Tiger verb, `Human.playSimpleGame`, has been extended to include progressive games. See About the Human.playSimpleGame Tiger Verb for more information.

- The Tiger verb attribute, *tiger:duration*, has been enhanced. See About the tiger:duration Attribute for more information.

Corrections

- Wager categories are now displayed for game play devices in the SmartEGM Data Model Viewer.
- Game Denominations are now displayed for each gamePlay device in the SmartEGM Data Model Viewer.
- RST now returns a G2S_APX001 (Invalid Host Identifier) error when an invalid host ID is received.
- The build date on the "About RST" screen is now updated whenever a build occurs.

About the Human.playSimpleGame Tiger Verb

The `Human.playSimpleGame` verb simulates the play of a simple, non-central determination game, a secondary game, and/or a progressive game. Progressive game play attributes appear in **bold** in the XML Representation Summary below. A sample code snippet for a progressive win appears in the "Examples" section.

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:credits-to-wager-cashable	=xs:int	0	Number of cashable credits wagered.
tiger:credits-to-wager-non-cashable	=xs:int	0	Number of non-cashable credits wagered.
tiger:credits-to-wager-promo	=xs:int	0	Number of promotional credits wagered.
tiger:denom-id	=xs:long	none	Denomination to be wagered.
tiger:device-id	=xs:int	-2	Device identifier. "-2" indicates that the first available device ID will be used.
tiger:primary-win	=xs:int	none	Number of credits awarded for the handle pull.
tiger:remote-key-off-timeout	=xs:int	60000	Indicates how long the script waits for the remote keyoff before timing out.
tiger:play-secondary-game-count	=xs:int	0	Number of double-or-nothing secondary games to play, if the primary-win is greater than zero (0). The first secondary game takes the primary-win value as the amount wagered. For all games except the last secondary game, the secondary amount won equals twice the amount wagered.
tiger:win-final-secondary-game	=xs:boolean	false	Indicates whether the final secondary game is a win or loss. If this attribute is set to "false," the final win equals zero (0).
tiger:progressive-hit	**=xs:Boolean**	**False**	**Indicates whether game play generates a progressive hit.**
tiger:win-level-index	**=xs:int**	**-1**	**Defines the win-level index for the progressive hit.**
tiger:win-to-handpay	=xs:boolean	false	Indicates whether the game win should go to a handpay.
tiger:handpay-action	=("HANDPAY" | "CREDIT" | "VOUCHER" | "WAT" | "REMOTE")	HANDPAY	Indicates how the handpay is paid.

Examples

- Plays a simple game on Game Play device 7, where two $0.25 cashable credits are wagered. A progressive win is hit for win level index 1. One thousand $0.25 cashable credits are won. A handpay is generated and paid by voucher.

```
<tiger:Human.playSimpleGame tiger:device-id="7" tiger:denom-id="25000"
tiger:credits-to-wager-cashable="2" tiger:progressive-hit=”true” tiger:
tiger:win-level-index=”1” tiger:primary-win="1000" handpay="true"
tiger:handpay-action="VOUCHER"/>
```

- Plays a simple game on Game Play device 7, where two $0.25 cashable credits are wagered and one $0.25 cashable credit is won. The game play causes a handpay event that is paid to a voucher.

```
<tiger:Human.playSimpleGame tiger:device-id="7" tiger:denom-id="25000"
tiger:primary-win="1" tiger:credits-to-wager-cashable="2" tiger:win-to-
handpay="true" tiger:handpay-action="VOUCHER"/>
```

- Plays a simple game on Game Play device 7, where two $0.25 cashable credits are wagered and one $0.25 cashable credit is won. The game play causes a handpay event. The game play device waits until a remote keyoff event is received from the host. If the remote keyoff is not received in 10 minutes, the `Human.playSimpleGame` verb fails.

```
<tiger:Human.playSimpleGame tiger:device-id="7" tiger:denom-id="25000"
tiger:primary-win="1" tiger:credits-to-wager-cashable="2" tiger:win-to-
handpay="true" tiger:handpay-action="REMOTE" tiger:remote-key-off-
timeout="600000"/>
```

About the tiger:duration Attribute

The *tiger:duration* attribute can be used in the following ways:

- as a constant number of iterations
- as a random number of iterations
- as an elapsed time

Constant Number of Iterations

If the value of the attribute is a string that defines a positive integer value (*x*), the loop is executed exactly *x* iterations. This use of the *tiger:duration* attribute is exactly the same as using the *iterations* attribute except the value of the attribute is a string and not an integer.

Example: Run the script 200 times.

```
<tiger:repeat tiger:duration="'200'" />
```

Random Number of Iterations

If the value of the attribute is a string of the pattern [low, high], the loop is executed a random number of times in the range [low, high].

If the value of the attribute is a string of the pattern [high], the loop is executed a random number of times in the range [1, high].

Example: Run the script, randomly, between 10 and 20 times.

```
<tiger:repeat tiger:duration="'[10, 20]'" />
```

Elapsed Time

This attribute specifies duration in terms of elapsed time. The value of the attribute is a string value that defines the amount of time to execute the body of the loop.

The value of the attribute must be a relative time value, in the format **D HH:MM:SS**, where:

- D is the number of days (must be at least one digit)
- HH is the number of hours (must be two digits)
- MM is the number of minutes (must be two digits)
- SS is the number of seconds (must be two digits)

The loop iterates until the time duration has elapsed. The actual number of iterations processed is dependent on what the script does and how fast the script executes on the host machine.

Example: Run the script as many times as possible during a 10-minute period.

```
<tiger:repeat tiger:duration="'0 00:10:00'" />
```

Note that the attribute content in each example is in single quotes. Single quotes allow you to use a parameter name instead of a string constant.

About the tiger:DataModel.snapshot Verb

The `tiger:DataModel.snapshot` verb causes the SmartEGM to take a snapshot of the data model and store it under the given name.

This verb may be included in the following elements: `tiger:action`, `tiger:catch`, `tiger:else`, `tiger:repeat`, `tiger:then`, `tiger:tiger`, `tiger:try`

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:name	xs:string	none	Name of the snapshot. This attribute is required for this verb.

Examples

This SmartEGM stores the current data model to a snapshot with the name of "shapshot1." The snapshot can be viewed through the Compare Data Model tab under the SmartEGM Data Model Viewer.

```
<tiger:DataModel.snapshot tiger:name="snapshot1"/>
```

Take a snapshot of the data model, and name it “start.” Run the script. Then, take another snapshot of the data model, and name it “end.”

```
<tiger:DataModel.snapshot tiger:name="start"/>
    <tiger:repeat tiger:iterations="loopDuration + 1">
        <tiger:Human.insertID tiger:device-id="1" tiger:id-number="12345678" />
        <tiger:Human.insertNotes tiger:device-id="1" tiger:currency-id="USD"
tiger:denom-id="100000"
            tiger:note-action="DROP" />
        <tiger:Human.insertCoins tiger:device-id="1" tiger:currency-id="USD"
tiger:denom-id="10000"
            tiger:coin-action="HOPPER" tiger:count="4" />
        <tiger:Human.playSimpleGame tiger:device-id="1" tiger:denom-id="1000"
tiger:primary-win="25000"
            tiger:credits-to-wager-cashable="5" />
        <tiger:Human.createVoucher tiger:device-id="1" tiger:credit-type="CASHABLE" />
        <tiger:Human.removeID tiger:device-id="1" />
        <tiger:DataModel.waitForEventQueueToDrain tiger:timeout="600000" />
    </tiger:repeat>
    <tiger:DataModel.snapshot tiger:name="end"/>
```

Using Tiger Scripts with an ID Database

To use ID database Tiger verbs – `Human.insertIDFromDatabase` and `Human.insertIDToDatabase` – you must create a file of player IDs. These IDs should correspond to the player IDs in your host system.

1. Open an XML file (using either an XML editor or by saving a text file with an .xml extension).

2. **Save** the file.

   If you are using a RadBlue **smartegm-example-database-00X.xml** Tiger script, type **id-example.xml** for the file name. Otherwise, the file name should reflect the value of the *database-name* attribute in the ID database Tiger verbs, in the format **id-[database-name].xml**.

3. Save the file to the following location: [**productDirectory**] > **radblue** > **gsa** > **script** > **conf**

4. Enter the following text, substituting the ID numbers for your host's player IDs:

   ```
   <id-database>
           <id number="22222222" lock="false"/>
           <id number="88888888" lock="false"/>
           <id number="12345678" lock="false"/>
           <id number="11111111" lock="false"/>
           <id number="99999999" lock="false"/>
   </id-database>
   ```

   The player ID numbers used in this example are valid if you are using the G2S Scope (RGS).

5. For each *id number*, enter "false" for the *lock* attribute. The *lock* attribute allows you to indicate that a player ID is *in-use* (inserted into an EGM). The *lock* attribute is automatically set to "true" when an ID is in use.

6. **Save** and **close** the file.

Using Tiger Scripts with a Voucher Database

To use voucher database Tiger verbs – `Human.insertVoucherFromDatabase` and `Human.createVoucherToDatabase` – you must create a file of voucher numbers. These 18-digit numbers should correspond to the voucher numbers in your host system.

If you are using a RadBlue **smartegm-example-database-00X.xml** Tiger script, a **voucher-example.xml** voucher database file is automatically created and populated with voucher numbers.

If you want to use voucher numbers from your host system, use the following procedure:

1. Open an XML file (using either an XML editor or by saving a text file with an .xml extension).

2. **Name** the file.

   The file name should reflect the value of the *database-name* attribute in the voucher database Tiger verbs, in the format **id-**[***database-name***]**.xml**. When you run your custom Tiger script, the voucher numbers from the specified database will be used.

3. **Save** the file to the following location: [**productDirectory**] > **radblue** > **gsa** > **script** > **conf**

4. Enter the following text, substituting the voucher numbers for voucher numbers in your host system's voucher database:

   ```
   <voucher-database>
           <id number= "300011224778107571”/>
           <id number= "300021224778107572”/>
           <id number= "300031224778107573”/>
           <id number= "300041224778107574”/>
           <id number= "300051224778107575”/>
   </voucher-database>
   ```

5. **Save** and **close** the file.

ID Database and Voucher Database Tiger Verbs

Human.insertIDFromDatabase

The `Human.insertIDFromDatabase` verb simulates the insertion of an ID into an ID reader. The ID number is read from the specified XML database.

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:database-name	= xs:string		Name of id database.
tiger:device-id	= xs:int	-2	ID reader to use.
tiger:lock-id	= xs:boolean	true	Indicates whether the ID record is to be locked while in-use.

Example

Using the first ID Reader device insert the ID number read from the **id-goldCards.xml** database. The ID number will be locked in the database, so that it cannot be reused until it is removed from the ID reader.

```
<tiger:Human.insertID tiger:device-id="-2" tiger:database-name="goldCards"
tiger:lock="true"/>
```

Human.removeIDToDatabase

The `Human.removeIDToDatabase` verb simulates the removal of an ID from an ID reader. The ID number that is removed is unlocked if it was locked when it was inserted.

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:device-id	= xs:int	-2	ID reader device identifier.

Example

Using the first ID Reader device, remove the current ID and release it back to the **id-goldCards.xml** database.

```
<tiger:Human.removeIDToDatabase tiger:device-id="-2" />
```

Human.insertVoucherFromDatabase

The `Human.insertVoucherFromDatabase` verb simulates the insertion of a voucher. The voucher validation ID is randomly selected (and removed) from the specified database.

XML Representation Summary

Attribute	Restrictions	Default	Description
Tiger:action	= ("DROP" | "REJECT" | "RETURN")	"DROP"	Indicates action taken at the voucher's insertion.
tiger:database-name	= xs:string		Name of database to get voucher from.
tiger:device-id	= xs:int	"-2"	Voucher device identifier.
id-reader-device-id	= xs:int	"-2"	ID reader device identifier.
note-acceptor-device-id	= xs:int	"-2"	Note acceptor device identifier.

Example

Using the first voucher device, insert a voucher from the **voucher-example.xml** database.

```
<tiger:Human.insertVoucherFromDatabase tiger:database-name="example"/>
```

Human.createVoucherToDatabase

The `Human.createVoucherToDatabase` verb simulates the creation of a voucher. The voucher's validation number is then stored in a database for later processing.

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:create-type	= ("CASHABLE" | "PROMO" | "NON_CASHABLE")	"CASHABLE"	Credit meter to use
tiger:database-name	= xs:string		Name of database to insert voucher into.
tiger:device-id	= xs:int	"-2"	Voucher device identifier.
id-reader-device-id	= xs:int	"-2"	ID reader device identifier.

Example

Using the first voucher device, take all of the current credits off of the cashable credit meter and create a voucher using that value. The voucher's validation number is stored in the **voucher-example.xml** database.

```
<tiger:Human.createVoucherToDatabase tiger:device-id="-2" tiger:credit-
type="CASHABLE" tiger:database-name="example"/>
```

if-voucher-available

The `if-voucher-available` verb allows you to have conditional logic that executes if there is a voucher available in the specified database. If there is a voucher available, the verbs in the "then" block are executed. If there are no vouchers available in the database, the verbs in the "else" block are executed.

XML Representation Summary

Attribute	Restrictions	Default	Description
tiger:database-name	= xs:string		Name of database to insert voucher into.

Example

If there is a voucher available in the **voucher-initial-load.xml** database, insert a voucher from that database, otherwise do nothing.

```
<tiger:if-voucher-available tiger:database-name="initial-load">
  <tiger:then>
   <tiger:insertVoucherFromDabase database-name="initial-load" />
  </tiger:then>
</tiger:if-voucher-available>
```

About the Example Tiger Scripts

Example Tiger scripts containing the new Tiger verbs are located in the following directory: [**productDirectory**] > **radblue** > **gsa** > **scripts** > **smart-egm**

Example Tiger scripts can be used as templates for creating custom Tiger scripts.

File Name: smartegm-example-databases-001.xml
This Tiger script:

1. verifies that the ID reader, voucher, note acceptor, and communications devices exist and are enabled (waiting up to five minutes for each to be enabled).
2. inserts a player ID from the **id-example.xml** database.
3. inserts five one-dollar notes.
4. cashes out the credit meter to a voucher, which is inserted into the **voucher-example.xml** database.
5. inserts voucher from the voucher database using the default voucher and note acceptor devices.
6. removes ID to the ID database.
7. repeats steps 2 through 6 ten times.

File Name: smartegm-example-databases-002.xml
This Tiger script:

1. verifies that the ID reader, voucher, note acceptor, and communications devices exist and are enabled (again waiting up to five minutes for each).
2. inserts a player ID from the **id-example.xml** database.
3. inserts 10 one-dollar notes.
4. cashes out the credit meter to a voucher, which is inserted into the **voucher-example.xml** database. Steps 2 and 3 are repeated 10 times to create 10 vouchers in the database.
5. Next, 10 vouchers are inserted from the voucher database, using the default note acceptor and voucher devices.
6. Finally, the player ID is removed from the ID reader, clearing the flag in the database.

File Name: smartegm-example-databases-003.xml
This Tiger script:

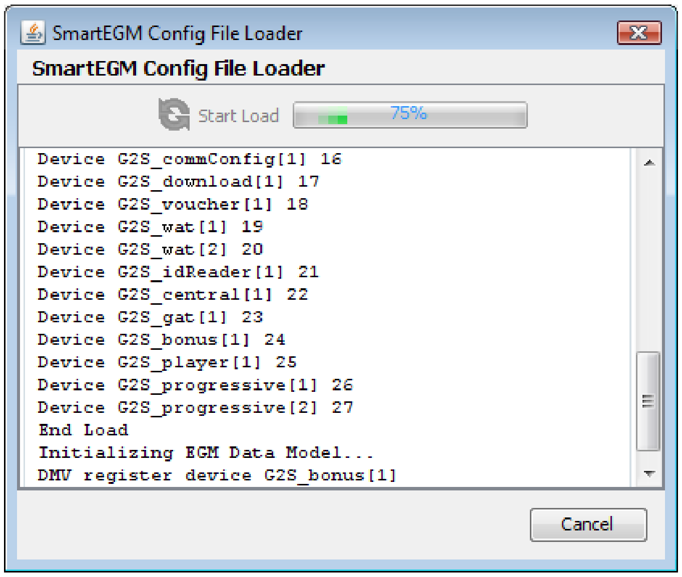
1. inserts a player ID from the ID database, but does not lock the ID.
2. removes the player ID from the ID reader.
3. repeats script ten times.

Version 1.6 [released: September 29, 2008]

High-Level Summary

In this release, we improved the efficiency of the SmartEGM and addressed a few minor issues.

Improvements



```
Device G2S_commConfig[1] 16
Device G2S_download[1] 17
Device G2S_voucher[1] 18
Device G2S_wat[1] 19
Device G2S_wat[2] 20
Device G2S_idReader[1] 21
Device G2S_central[1] 22
Device G2S_gat[1] 23
Device G2S_bonus[1] 24
Device G2S_player[1] 25
Device G2S_progressive[1] 26
Device G2S_progressive[2] 27
End Load
Initializing EGM Data Model...
DMV register device G2S_bonus[1]
```

- The SmartEGM has been modified to load configuration files more efficiently.
- The SmartEGM Configuration File Loader now updates while the SmartEGM configuration file is being loaded into the Data Model Viewer. A percentage appears on the status bar to let you see exactly where the file is in the load process.
- If XML content in a message is greater than 2MB, the following error message is logged as a warning in the Debug Log: "XML content is too large to store in SOAP transcript. Maximum length is 2MB. The first 1024 bytes:" (followed by the first 1024 bytes of the message).
- The Active Devices tab is obsolete, and it has been removed from the SmartEGM.

Corrections

- The SmartEGM has been modified to correctly handle an invalid *transferLocation* on `download.addPackage` commands. In accordance with the GSA G2S specification, a DLE103 error is returned.
- The cabinet G2S_egmPaidBonusNonWonAmt meter now updates correctly when a noncash bonus is awarded.
- The BNE104 (bonus paid event) is now generated when a bonus is paid to the credit meter.

Version 1.5 [released: September 2, 2008]

High-Level Summary

In version 1.5, we have updated the Multicast transport to reflect the changes made in the GSA Multicast Transport Protocol 1.0.7. We have added two new S2S desktops and corrected several issues.

Improvements

- Multicast has been implemented in accordance with the GSA Multicast Transport Protocol v1.0.7.
    - Test vectors have been implemented (as recently discussed in the GSA Transport Committee). Test vectors allow vendors to run test input against mtp schemas and verify that the results are the same. Our Multicast implementation has also been vetted by two major gaming manufacturers.
    - Multicast now uses a separate command ID sequence for Multicast messages.
    - New Multicast URI schemes have been added to RGS.

      com-gamingstandards-mtp://[ip-address]:[port] – unencrypted but authenticated, using UMAC.

      com-gamingstandards-mtps://[ip-address]:[port] – encrypted and authenticated, using UMAC-AE. (default)
- The SmartEGM now dynamically determines the value of attributes for `commsOnline` messages except for the *meterReset* attribute. The value of the *meterReset* attribute is based on the configuration file.
- Two new desktops have been created for RST – S2S_Edge and S2S_Central. The new desktops make it easier to use two versions of the tool as host and edge servers.
- The *timeToLive* attribute is automatically set to zero (0) on command responses and notifications. The *timeToLive* attribute is the number of milliseconds an endpoint should wait before ignoring a command. The *timeToLive* attribute remains the same on request messages.

Corrections

- The meter names for voucher devices have been corrected.
- If there is no event or meter subscription set when the `commsOnline` command is created, the SmartEGM now sets the *subscriptionLost* attribute properly.
- RST now supports XML escape sequences in the EGM ID.
- The SmartEGM Data Model Viewer (DMV) now displays the correct value of the *hostEnabled* attribute for all devices.
- The Transcript Control now accepts ASCII 7 (“tab”) characters in the XML payload.
- The SmartEGM now handles the noteAcceptor.getNotesAccepted command correctly.

Version 1.4 [released: June 30, 2008]

High-Level Summary

In release 1.4, we have made several usability improvements to RST in addition to corrections to the workings of the commConfig class.

Improvements

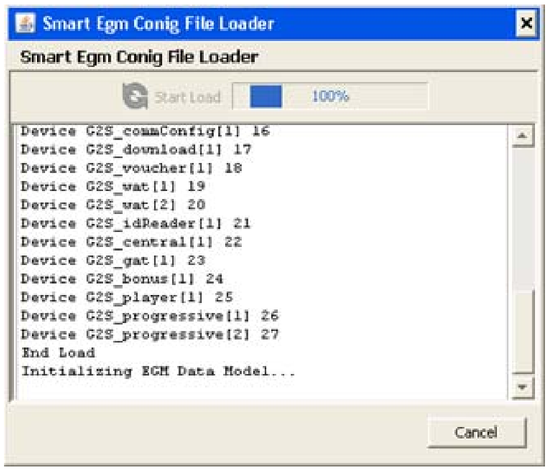
- A new SmartEGM configuration file was added that causes the SmartEGM to correctly send all messages through the RadBlue Protocol Analyzer (**smartegm-config-rpa.xml**).

- If you are using RadBlue tools on a Linux PC that is not connected to a network, you must issue the following command to allow the operating system to recognize the network device: **/sbin/ifconfig <deviceID> up**

  <deviceID> should be replaced with the network adapter containing the MAC address that was used to generate the license file. The most common device ID is **eth0**, making the command: **/sbin/ifconfig eth0 up**

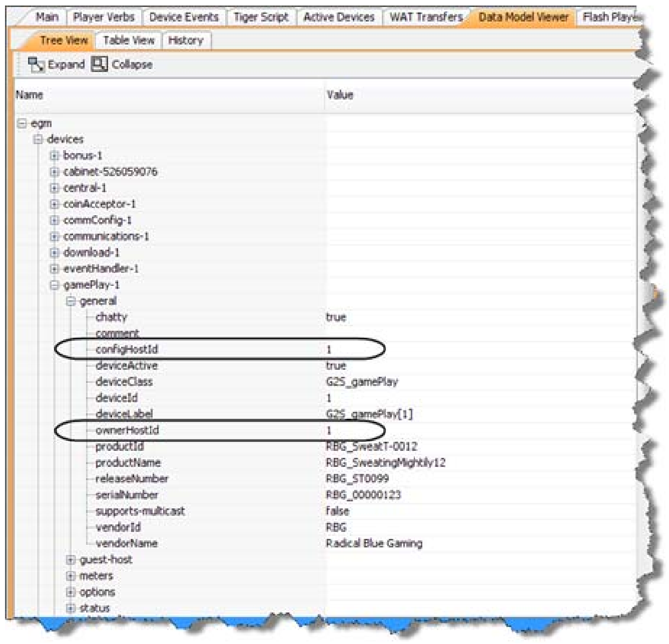
  Updated Linux installation instructions are available here: http://www.radblue.com/documentation/linux_readme.txt

- A progress screen has been added to the SmartEGM so the user can see the status of the configuration file load operation.



```
Device G2S_commConfig[1] 16
Device G2S_download[1] 17
Device G2S_voucher[1] 18
Device G2S_wat[1] 19
Device G2S_wat[2] 20
Device G2S_idReader[1] 21
Device G2S_central[1] 22
Device G2S_gat[1] 23
Device G2S_bonus[1] 24
Device G2S_player[1] 25
Device G2S_progressive[1] 26
Device G2S_progressive[2] 27
End Load
Initializing EGM Data Model...
```

- The following attribute names have been changed for clarity in the SmartEGM Data Model Viewer in the **general** group for each device:



Name	Value
egm	
devices	
bonus-1	
cabinet-526059076	
central-1	
coinAcceptor-1	
commConfig-1	
communications-1	
download-1	
eventHandler-1	
gamePlay-1	
general	
chatty	true
comment	
configHostId	1
deviceActive	true
deviceClass	G2S_gamePlay
deviceId	1
deviceLabel	G2S_gamePlay[1]
ownerHostId	1
productId	RBG_SweatT-0012
productName	RBG_SweatingMightily12
releaseNumber	RBG_ST0099
serialNumber	RBG_00000123
supports-multicast	false
vendorId	RBG
vendorName	Radical Blue Gaming
guest-host	
meters	
options	
status	

  - o *configId* has been changed to *configHostId*
  - o *ownerId* has been changed to *ownerHostId*

Corrections

- The SmartEGM has been updated to handle denomination ID (*denomId*) values up to 9,999,999,999.99.999, as defined by version 1.0.3 of the G2S protocol.

- The RadBlue package files available on our web site (package-1.zip and package-2.zip), did not work properly with all Zip utilities. The new package versions work correctly.

- Some shortcomings were discovered in the way that the SmartEGM was handling associated data in event subscriptions. These issues have been corrected.

- The SmartEGM was always sending an `eventReport` as a G2S_request. If persist equals "false," the event is now sent as a G2S_notfication.

- The SmartEGM now correctly handles anonymous FTP transfers when attempting to download a package.

- The following updates have been made to the commConfig implementation:
  - The `setCommConfig` command handler has been modified so that the SmartEGM communicates with affected hosts when the configuration of a device is changed.
  - The SmartEGM now correctly handles the changing of a device's owner or configuration host.
  - A `commsClosing` command is now sent when the communications channel restarts. It is followed automatically with a `commsOnline` command. The `commConfig.setCommChange` is an example of when this reboot sequence occurs.

commConfig Implementation Note

The default for the *canModRemote* attribute in the *hostIndex* table is always true. This attribute **must** be true for the EGM (*hostid*=0 and *hostIndex*=0), in order to be able to change the configuration host or owner host of a device from a zero (0) to a non-zero number.

Version 1.3 [released: May 27, 2008]

High-Level Summary

In release 1.3, the progressive class has been added, including two new player verbs, progressive data model information, and a new Progressives tab. A new security (SSL) options have been added to the Configure screen.

New Features

- RST now supports the progressive class. See Using Progressive.
    - A new Tiger/XML verb, `tiger:Progressive.getHostInfo`, has been added to request progressive host information.
    - **Get Progressive Host Info** and **Play Progressive Game** buttons have been added to the SmartEGM Player Verbs tab.
    - The following cabinet attributes determine whether a win is paid to the credit meter or as a handpay:

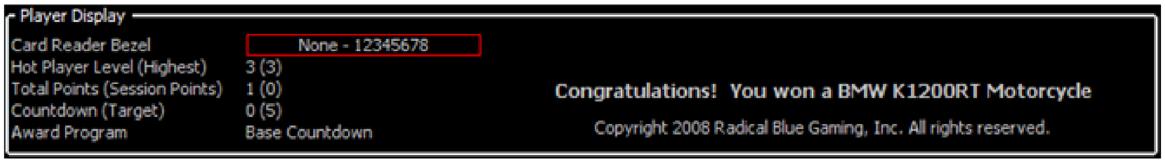
Attribute	Description
largeWinLimit	Maximum win payable by the EGM.
maxCreditMeter	Maximum value permitted on the credit meter.
maxHopperPayOut	Maximum value that can be paid from the hopper before a handpay, voucher, or WAT transaction is required.
splitPayOut	When *maxHopperPayOut* is reached, indicates whether the *maxHopperPayOut* amount should be paid from the hopper.

      If the *paymentMethod* attribute in the `setProgressiveWin` command sent by the progressive host is **payHandpay**, the SmartEGM performs a local handpay.

      If the progressive win is greater than the cabinet's *largeWinLimit* profile setting, the SmartEGM also performs a local handpay.

- A **Progressives** tab has been added to the SmartEGM. The Progressives tab displays progressive information. The `setProgressiveValue` command updates this screen.

- Progressive information has been added to the SmartEGM Player Display.



Player Display

Card Reader Bezel	None - 12345678
Hot Player Level (Highest)	3 (3)
Total Points (Session Points)	1 (0)
Countdown (Target)	0 (5)
Award Program	Base Countdown

Congratulations! You won a BMW K1200RT Motorcycle

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

When a `setProgressiveWin` command is received by RST, the *textMessage* attribute is displayed in the Player Display.

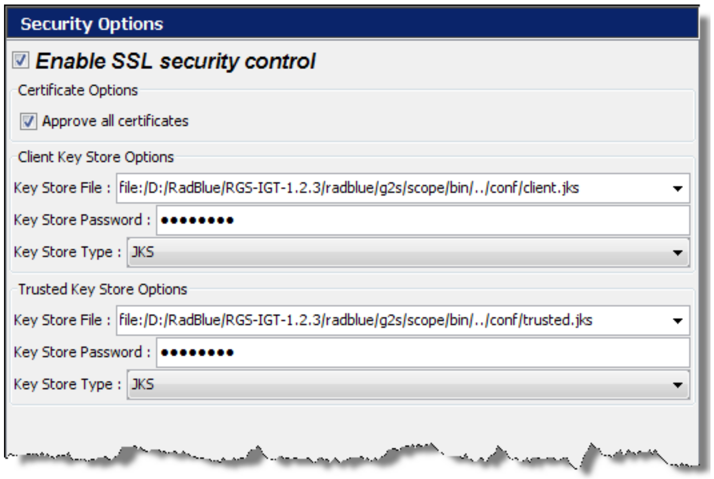
- The following option have been added to **Configure** > **Engine Options**:

  Filter G2S Set Progressive Values from Transcript – Select to exclude G2S `setProgressiveValue` messages from the Transcript and Debug Log. Note that `setProgressiveValueAck` messages are not filtered with this option. To filter the progressive ACK and the G2S ACK, use the Filter option available in the Transcript and Debug Log.

o RST can be configured to expect the progressive host to regularly send `setProgValue` messages, or that feature can be disabled completely. To configure the RST's expectations, you can set the *noProgInfo* attribute using the `setOptionChange` command or you can edit the progressive device parameters in the **smartegm-config.xml** file. See Using Progressive for information on editing the **smartegm-config.xml** file. The default value is 0, meaning the `setProgressiveValue` commands are not expected).

Improvements

- A **Security Options** screen has been added to the **Configure** option on the menu bar. From this screen, you can configure Secure Socket Layer (SSL) encryption information for the application.



Security Options

☑ Enable SSL security control

Certificate Options

☑ Approve all certificates

Client Key Store Options

Key Store File : file:/D:/RadBlue/RGS-IGT-1.2.3/radblue/g2s/scope/bin/../conf/client.jks

Key Store Password : ••••••••

Key Store Type : JKS

Trusted Key Store Options

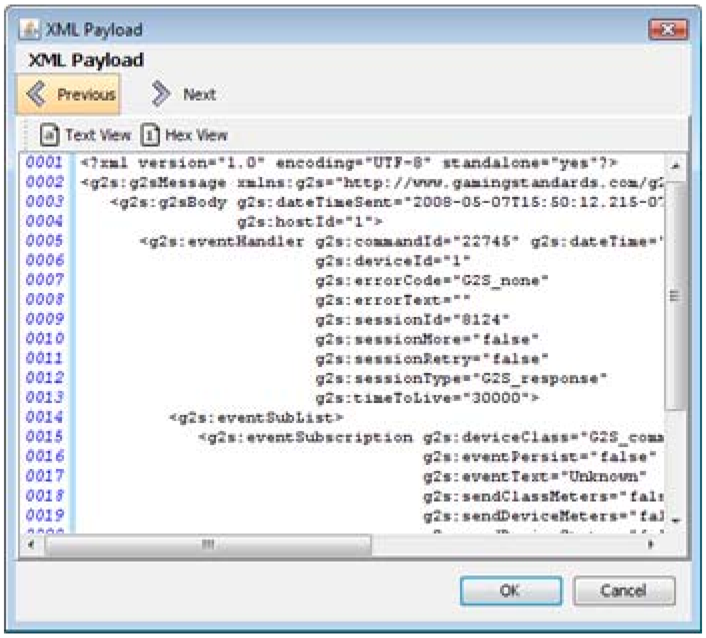
Key Store File : file:/D:/RadBlue/RGS-IGT-1.2.3/radblue/g2s/scope/bin/../conf/trusted.jks

Key Store Password : ••••••••

Key Store Type : JKS

- Select **Enable SSL security control** to enable encryption for the application.
- Select **Approve all certificates** if you want to use SSL encryption, but are not concerned with the validity of the certificate authority.
- Enter the **Key Store Options** for the client and the certificate authority.

- You can now navigate through multiple **XML Payload** screens, which can be accessed by double-clicking a message entry in the **Transcript Control** object.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<g2s:g2sMessage xmlns:g2s="http://www.gamingstandards.com/g2
   <g2s:g2sBody g2s:dateTimeSent="2008-05-07T15:50:12.215-07
                g2s:hostId="1">
      <g2s:eventHandler g2s:commandId="22745" g2s:dateTime="
                        g2s:deviceId="1"
                        g2s:errorCode="G2S_none"
                        g2s:errorText=""
                        g2s:sessionId="8124"
                        g2s:sessionMore="false"
                        g2s:sessionRetry="false"
                        g2s:sessionType="G2S_response"
                        g2s:timeToLive="30000">
         <g2s:eventSubList>
            <g2s:eventSubscription g2s:deviceClass="G2S_comm
                                   g2s:eventPersist="false"
                                   g2s:eventText="Unknown"
                                   g2s:sendClassMeters="fal
                                   g2s:sendDeviceMeters="fal
```

Click the **Previous** and **Next** buttons to move between entries without exiting the XML Payload window.

If you have applied sorting options to the displayed Transcript information, the navigation will follow the sort order of the main display window.

Corrections

- A SmartEGM issue with the `communications.getDescriptor` command has been corrected. Previously, the "include" flags were being handled incorrectly.

Using Progressive

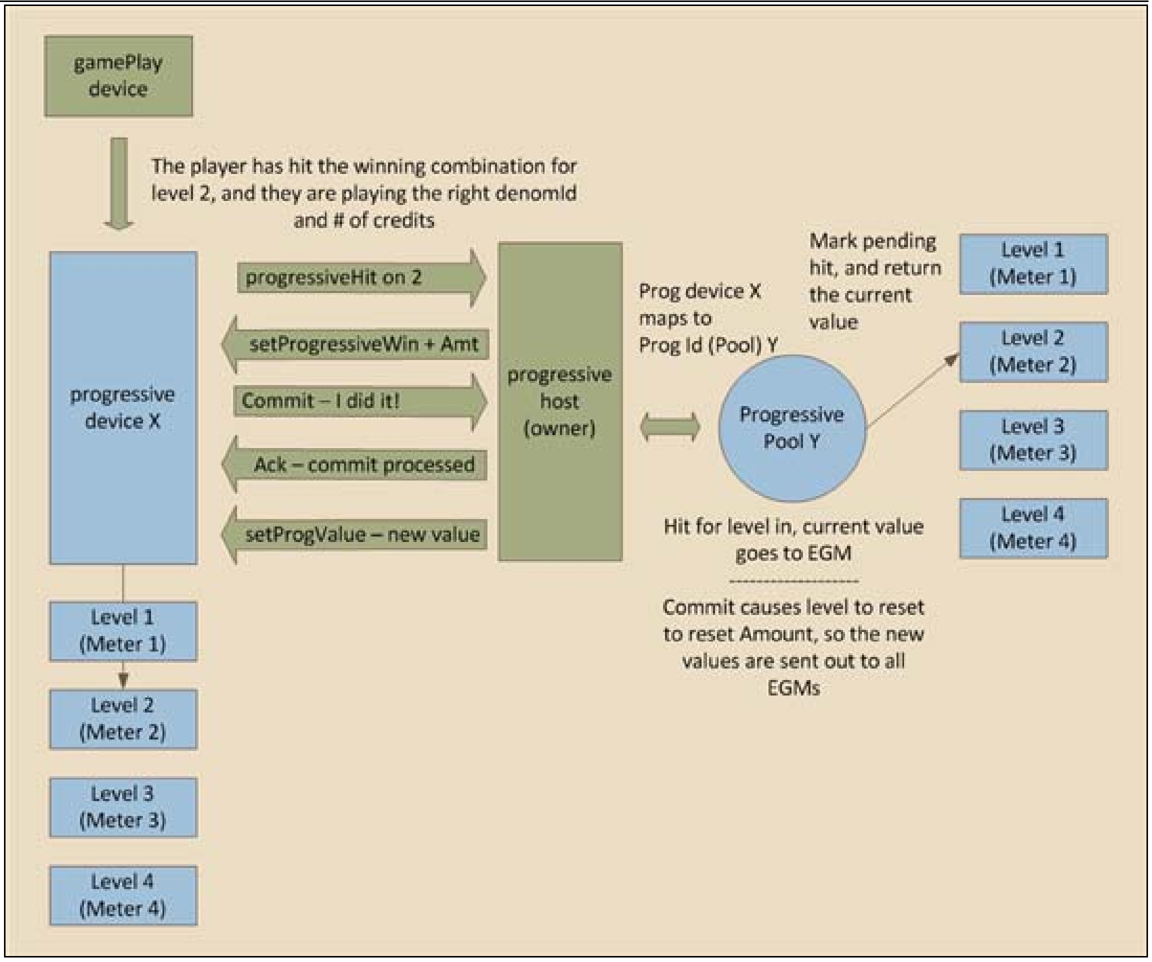
A progressive device is owned by a progressive host, which identifies the *progressive ID*, a group of progressive levels or meters, and the levels in that group supported by the EGM. There is exactly one progressive ID for each progressive device, and one or more progressive levels for each progressive device.

Each progressive level can be hit by one or more win-level, in one or more gamePlay devices in the EGM. The win-level is identified by a *win level index* (a unique identifier that represents a combination of reel positions, for example, a full house or royal flush on a poker game).

The mapping between progressive level and win level in a game is:

progressiveId + progressiveLevel ⬄ gamePlay device + winLevelIndex + wager (# credits * denom)

Sample Progressive Flow


gamePlay device
The player has hit the winning combination for level 2, and they are playing the right denomId and # of credits
progressive device X
Level 1 (Meter 1)
Level 2 (Meter 2)
Level 3 (Meter 3)
Level 4 (Meter 4)
progressiveHit on 2
setProgressiveWin + Amt
Commit – I did it!
Ack – commit processed
setProgValue – new value
progressive host (owner)
Prog device X maps to Prog Id (Pool) Y
Mark pending hit, and return the current value
Progressive Pool Y
Hit for level in, current value goes to EGM
Commit causes level to reset to reset Amount, so the new values are sent out to all EGMs
Level 1 (Meter 1)
Level 2 (Meter 2)
Level 3 (Meter 3)
Level 4 (Meter 4)

The event that updates the host's progressive database is G2S_PGE101 (Progressive Money Wagered). If this event is not generated by the EGM (not supported or not subscribed to by the host), the host's progressive values won't update when a game is played. The `setProgressiveValue` command is then used by the host to send the updated progressive values back to the EGM.

Configuring Progressives in the SmartEGM

1. Navigate to: **..\radblue\gsa\script\smart-conf\smart-egm**.
2. Right-click **smartegm-config.xml**, and select **Edit**.

To configure progressives in RadBlue tools, you first need to configure the various win levels supported by the gamePlay device. These values are typically characteristics of the game, so they are not configurable through the G2S optionConfig class. The following excerpt is from gamePlay device 1 in the sample **smartegm-config.xml** file distributed with the tools:

gamePlay Device Section

```
<edm:win-levels>
      <edm:win-level edm:index="1" edm:win-level-combo="1 Troll"
            edm:progressive-allowed="true" edm:win-level-odds="1" />
      <edm:win-level edm:index="12" edm:win-level-combo="Lots o Trolls"
            edm:progressive-allowed="true" edm:win-level-odds="21" />
</edm:win-levels>
```

Next, you need to configure the progressive device to tie a progressive level (meter) to a particular gamePlay device and win-level. This is configurable through G2S and can be done using the `setOptionList` command, or by editing the **smartegm-config.xml** file for the progressive device you want to hit when the appropriate winning combination is hit in the gamePlay device. A progressive data table is used in the progressive device to define the relationship of the progressive level and the gamePlay win level:

Progressive Device Section – Progressive Data Table

```
<edm:option edm:option-id="G2S_progDataTable">
  <edm:parameters-table>
     <edm:parameters edm:param-id="G2S_progData">
      <edm:parameter edm:param-id="G2S_denomId">100000</edm:parameter>
      <edm:parameter edm:param-id="G2S_themeId">
            RBG_sweatyTrolls</edm:parameter>
      <edm:parameter edm:param-id="G2S_paytableId">RBG_92</edm:parameter>
      <edm:parameter edm:param-id="G2S_numberOfCredits">3</edm:parameter>
      <edm:parameter edm:param-id="G2S_levelId">1</edm:parameter>
      <edm:parameter edm:param-id="G2S_gamePlayId">1</edm:parameter>
      <edm:parameter edm:param-id="G2S_winLevelIndex">1</edm:parameter>
     </edm:parameters>
```

To modify the amount of time that the SmartEGM waits for a `setProgressiveValue` message from the progressive host (if at all), select the **G2S_noProgInfo** parameter and change the value to the amount of time, in milliseconds, that you want RST to wait for interval messages from the host. This time should be greater than the interval setting in the host. Zero (0) disables this feature.

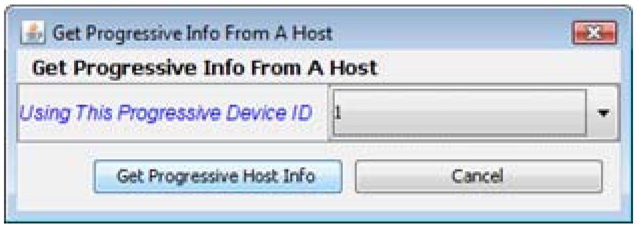
Progressive Device Section – Option Settings

```
  edm:option-settings>
    <edm:option-group edm:option-group-id="G2S_progressiveOptions"
 edm:option-group-name="G2S Progressive Options">
    <edm:option edm:option-id="G2S_protocolOptions">
    <edm:parameters edm:param-id="G2S_protocolParams">
    <edm:parameter edm:param-id="G2S_timeToLive">0</edm:parameter>
    <edm:parameter edm:param-id="G2S_requiredForPlay">false</edm:parameter>
    <edm:parameter edm:param-id="G2S_configurationId">0</edm:parameter>
    <edm:parameter edm:param-id="G2S_restartStatus">true</edm:parameter>
    <edm:parameter edm:param-id="G2S_useDefaultConfig">true</edm:parameter>
    <edm:parameter edm:param-id="G2S_progId">10</edm:parameter>
    <edm:parameter edm:param-id="G2S_noResponseTimer">30000</edm:parameter>
    <edm:parameter edm:param-id="G2S_noProgInfo">0</edm:parameter>
    </edm:parameters>
    </edm:option>
```

Get Progressive Information

This procedure allows you to query the progressive host to discover the progressive identifiers and levels that the host supports.

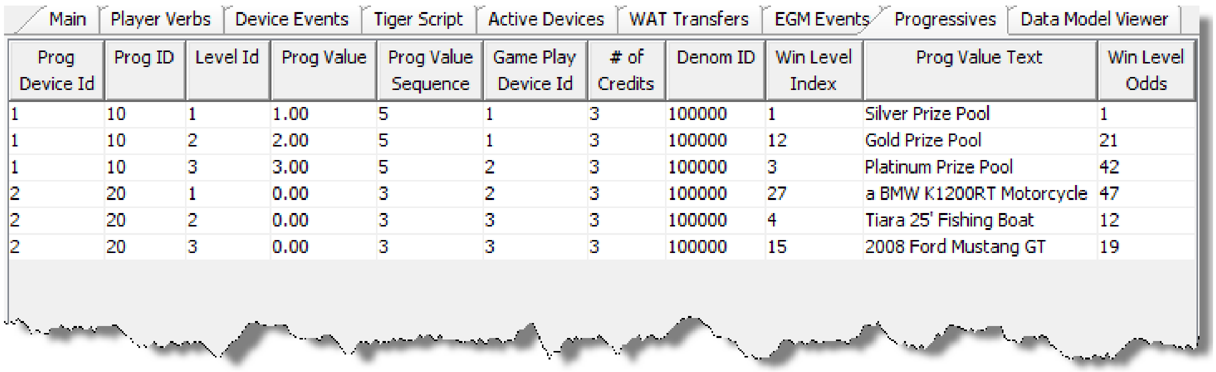
1. From the SmartEGM layout, select **Player Verbs**.
2. Click **Get Progressive Host Info**.


Get Progressive Info From A Host
Get Progressive Info From A Host
Using This Progressive Device ID
1
Get Progressive Host Info
Cancel

3. Click the drop-down arrow, and select the **Progressive Device ID** to use for the query.
4. Click **Get Progressive Host Info**.

View Progressive Information

To view progressive device information, click the new **Progressives** tab on the SmartEGM layout. This screen displays complete information about each progressive level, for each progressive device. This display is updated in real-time whenever a `setProgressiveValue` command is received from the host.



Prog Device Id	Prog ID	Level Id	Prog Value	Prog Value Sequence	Game Play Device Id	# of Credits	Denom ID	Win Level Index	Prog Value Text	Win Level Odds
1	10	1	1.00	5	1	3	100000	1	Silver Prize Pool	1
1	10	2	2.00	5	1	3	100000	12	Gold Prize Pool	21
1	10	3	3.00	5	2	3	100000	3	Platinum Prize Pool	42
2	20	1	0.00	3	2	3	100000	27	a BMW K1200RT Motorcycle	47
2	20	2	0.00	3	3	3	100000	4	Tiara 25' Fishing Boat	12
2	20	3	0.00	3	3	3	100000	15	2008 Ford Mustang GT	19

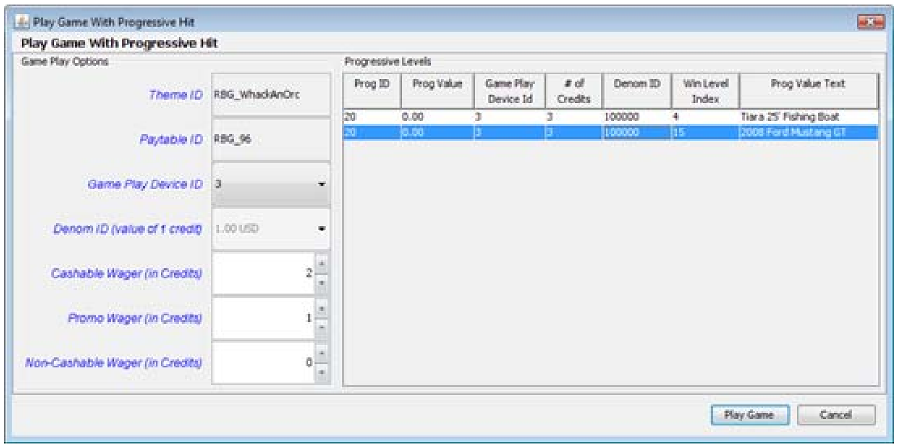
- **Prog Device ID** – Progressive device identifier.
- **Prog ID** – Progressive group identifier.
- **Level ID** – Identifier of progressive level within a progressive group.
- **Prog Value** – Current value of the progressive.
- **Prog Value Sequence** – Unique identifier set by the host in each `setProgressiveValue` command.
- **Game Play Device ID** – Identifier of game play device that ties to this progressive.
- **# of Credits** – Number of credits required to hit specified progressive.
- **Denom ID** – Denomination that must be played for the progressive.
- **Win Level Index** – Paytable win index that hits this progressive.
- **Win Level Odds** – Odds of hitting this index in the paytable.

View Progressive Settings for the EGM

1. From the SmartEGM layout, click **Data Model Viewer**.
2. Navigate to: **egm** > **devices** > **progressive-1** > **options** > **G2S_progressiveOptions**.
3. Click **G2S_progDataTable** to view information about the current progressive data table (progressive-level-to-gameplay-win-level linkages).
4. Click **G2S_protocolOptions** to view progressive option settings.

Play a Progressive Game

1. From the **Player Verbs** object, click **Play Progressive Game**.



Play Game With Progressive Hit

Game Play Options

Theme ID	RBG_WhackAnOrc
Paytable ID	RBG_96
Game Play Device ID	3
Denom ID (value of 1 credit)	1.00 USD
Cashable Wager (in Credits)	2
Promo Wager (in Credits)	1
Non-Cashable Wager (in Credits)	0

Progressive Levels

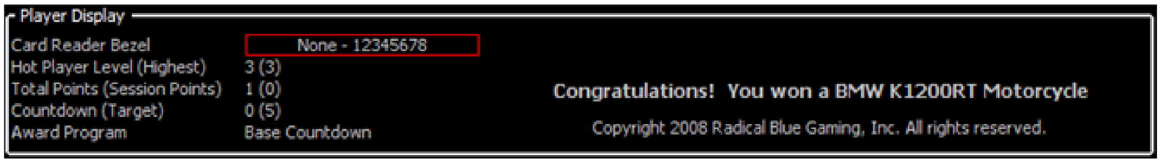
Prog ID	Prog Value	Game Play Device Id	# of Credits	Denom ID	Win Level Index	Prog Value Text
20	0.00	3	3	100000	4	Tiara 25' Fishing Boat
20	0.00	3	3	100000	15	2008 Ford Mustang GT

2. Under **Game Play Options**, click the **Game Play Device ID** drop-down arrow, and select the progressive device that you want to play.

3. Click the progressive you want to hit from the **Progressive Levels** section. The Progressive Levels section lists all available progressives for the selected progressive device. Note that the denomination (**Denom ID**) field changes to the required denomination for the progressive you selected.

4. Enter the number of credits you want wager, using the **Cashable Wager**, **Promo Wager** and **Non-Cashable Wager** combo boxes. You can allocate your wager in any combination of the three wager types, but the total wagered must equal the number of credits specified in the **# of Credits** field. If not, the **Play Game** button will not activate.

5. Click **Play Game**.
   After a progressive hit, the progressive is reset. You can see the reset on the Progressives tab. The progressive prize displays on the Player Display:



Player Display

Card Reader Bezel	None - 12345678
Hot Player Level (Highest)	3 (3)
Total Points (Session Points)	1 (0)
Countdown (Target)	0 (5)
Award Program	Base Countdown

Congratulations! You won a BMW K1200RT Motorcycle

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

Version 1.2 [released: April 25, 2008]

High-Level Summary

In this newest release, we added support for the G2S player class, a new EGM Transcript Analysis report, and made even more improvements to the rest of the product.

New Features

2. RST now supports the G2S player class.

    a. A Player Display panel has been added to the SmartEGM. See About the Player Display.

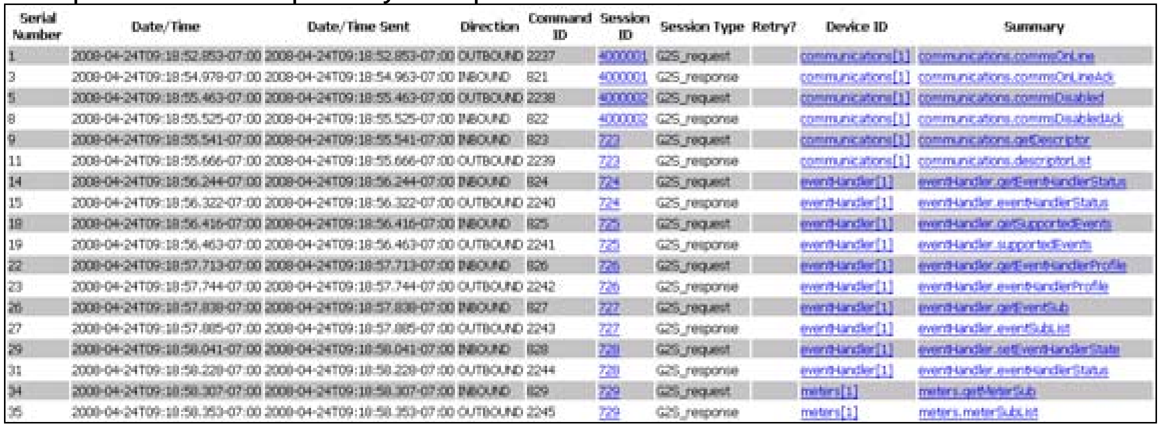
    b. Hot Player Support – Hot Player support is provided for carded and uncarded players using the player class parameters. The appropriate events are sent based on whether the player is using an ID. As per the specification, at the end of the hot player period, the hot player level resets to zero (0). Hot player determination can be based on a single meter or an RPN expression.

    c. Player Rating Support – Player ratings, metering, event, messages, and countdowns are supported, including player and generic override point calculation formulas (using a single meter or an RPN expression for the basis).

    d. SmartEGM Configuration File Changes – A new player attribute, *thread-sleep-timer*, has been added to the **smartegm-config.xml file**. This attribute controls how often the program checks whether the hot player time period has expired. Once the hot player time period has expired, the timer is reset. The default time is 60000 milliseconds (one minute).

e. A new **EGM Transcript Analysis report** has been added to the Transcript Control object. The EGM Transcript Analysis report provides information about messages that were sent from and received by the application for the period requested. See the *EGM Transcript Analysis Report* for more information.

Sample EGM Transcript Analysis Report



Serial Number	Date/Time	Date/Time Sent	Direction	Command ID	Session ID	Session Type	Retry?	Device ID	Summary
1	2008-04-24T09:18:52.853-07:00	2008-04-24T09:18:52.853-07:00	OUTBOUND	2237	4000001	G2S_request		communications[1]	communications.commsOnLine
3	2008-04-24T09:18:54.978-07:00	2008-04-24T09:18:54.963-07:00	INBOUND	821	4000001	G2S_response		communications[1]	communications.commsOnLineAck
5	2008-04-24T09:18:55.463-07:00	2008-04-24T09:18:55.463-07:00	OUTBOUND	2238	4000002	G2S_request		communications[1]	communications.commsDisabled
8	2008-04-24T09:18:55.525-07:00	2008-04-24T09:18:55.525-07:00	INBOUND	822	4000002	G2S_response		communications[1]	communications.commsDisabledAck
9	2008-04-24T09:18:55.541-07:00	2008-04-24T09:18:55.541-07:00	INBOUND	823	723	G2S_request		communications[1]	communications.getDescriptor
11	2008-04-24T09:18:55.666-07:00	2008-04-24T09:18:55.666-07:00	OUTBOUND	2239	723	G2S_response		communications[1]	communications.descriptorList
14	2008-04-24T09:18:56.244-07:00	2008-04-24T09:18:56.244-07:00	INBOUND	824	724	G2S_request		eventHandler[1]	eventHandler.getEventHandlerStatus
15	2008-04-24T09:18:56.322-07:00	2008-04-24T09:18:56.322-07:00	OUTBOUND	2240	724	G2S_response		eventHandler[1]	eventHandler.eventHandlerStatus
18	2008-04-24T09:18:56.416-07:00	2008-04-24T09:18:56.416-07:00	INBOUND	825	725	G2S_request		eventHandler[1]	eventHandler.getSupportedEvents
19	2008-04-24T09:18:56.463-07:00	2008-04-24T09:18:56.463-07:00	OUTBOUND	2241	725	G2S_response		eventHandler[1]	eventHandler.supportedEvents
22	2008-04-24T09:18:57.713-07:00	2008-04-24T09:18:57.713-07:00	INBOUND	826	726	G2S_request		eventHandler[1]	eventHandler.getEventHandlerProfile
23	2008-04-24T09:18:57.744-07:00	2008-04-24T09:18:57.744-07:00	OUTBOUND	2242	726	G2S_response		eventHandler[1]	eventHandler.eventHandlerProfile
26	2008-04-24T09:18:57.838-07:00	2008-04-24T09:18:57.838-07:00	INBOUND	827	727	G2S_request		eventHandler[1]	eventHandler.getEventSub
27	2008-04-24T09:18:57.885-07:00	2008-04-24T09:18:57.885-07:00	OUTBOUND	2243	727	G2S_response		eventHandler[1]	eventHandler.eventSubList
29	2008-04-24T09:18:58.041-07:00	2008-04-24T09:18:58.041-07:00	INBOUND	828	728	G2S_request		eventHandler[1]	eventHandler.setEventHandlerState
31	2008-04-24T09:18:58.228-07:00	2008-04-24T09:18:58.228-07:00	OUTBOUND	2244	728	G2S_response		eventHandler[1]	eventHandler.eventHandlerStatus
34	2008-04-24T09:18:58.307-07:00	2008-04-24T09:18:58.307-07:00	INBOUND	829	729	G2S_request		meters[1]	meters.getMeterSub
35	2008-04-24T09:18:58.353-07:00	2008-04-24T09:18:58.353-07:00	OUTBOUND	2245	729	G2S_response		meters[1]	meters.meterSubList

Improvements

- The SmartEGM configuration file loader has been enhanced to perform additional semantic error checking on the contents of the file (looking for errors related to the host table, non-existent devices, and multiple instances of a single instance class).

- Several new Tiger/XML verbs have been added to the SmartHost Tiger/XML script. See New and Modified SmartHost Tiger/XML Verbs.

- The Device Manager has been reworked to be more efficient, resulting in a faster SmartEGM.

- If you add a non-RadBlue package to the SmartEGM and send a `gat.getComponentList` command, you will not see your package in the list. This is because the package does not have a validation algorithm assigned to it and G2S 1.0.3 does not allow components without at least one algorithm.

  When the SmartEGM drops your package from the `gat.componentList` this fact will be reported in the logger.

- The communications device in the **smartegm-config.xml** file has a new attribute, *date-time-delta*. This feature is useful for verifying that there is no drift between the clock on the RST PC and the clock on the G2S Host PC. This is particularly important when testing for environments without an NTP server.

  If the value is non-zero, this attribute defines the number of milliseconds that the *dateTime* value on the received G2S command can differ from the clock on the local computer. If the difference is greater than the *data-time-delta* value, a warning message is printed to the log panel.

  If the value of the *data-time-delta* is zero, this feature is disabled. The default is zero (0).

  Note that this feature should be disabled if RST is running under a heavy load. CPU starvation of the RST will generate false positives.

- In the **smartegm-config.xml** file, you can specify pre-loaded GAT components (modules and packages). However, the **smartegm-config.xml** file does not require any validation algorithms to be defined for a given component.

  RST now automatically adds all seven algorithms listed in G2S (MD5, CRC16, CRC32, SHA1, SHA256, SHA389 and SHA512) if no algorithms are specified in the configuration file.

- Added support for user name and password in the transferLocation URL, in the following format: **ftp://user:password@host:port//path**

  In the `addPackage` and `uploadPackage` commands, RST uses the user name and password in the following order:

  1. User name and password from the *transferParameters* attribute.
  2. User name and password from FTP URL.

  If neither of the above exists, RST does not use a user name or password.

- Added support for startOffset >= endOffset in the `gat.doVerification` command. The code now determines if the buffer to hash needs to wrap to the front of the buffer (startOffset >= endOffset). This allows the EGM to support gat.doVerfication as specified in the G2S protocol.

- A new attribute, *honor-time-to-live*, has been added to the communications device in the **smartegm-config.xml** file. If set to **true**, the SmartEGM compares the current time against the *dateTime* attribute as well as the *timeToLive* attribute in the G2S command. If the current time is after the specified time-to-live, the message has expired and an APX001 message is sent. Otherwise, the command is processed by the SmartEGM.

  If set to false (the default), the SmartEGM does not look at the *time-to-live* attribute.

- You now have the option to disable message validation in the SmartEGM. To disable message validation:

  1. Quit RST.
  2. Edit the **bin** > **rst-launcher.xml** file with a text editor.
  3. Change the following line:
     com.radblue.g2s.egm.datamodel.core.types.messages.validation=true
     to:
     com.radblue.g2s.egm.datamodel.core.types.messages.validation=false
  4. Save and close the file.
  5. Restart RST.

<u>Corrections</u>

- The Transcript database could not store *commandId* values greater than 32 bits. Sending one of our products a *commandId* greater than 32 bits would result in an exception being displayed in the Logger panel and the transcript entry would not be added to the database.

- Prior to 1.2 the SmartEGM would return the requested options regardless of the host access control rules specified in the G2S protocol. In particular, non-owner/non-guest/non-config hosts could retrieve the options, which is a violation of the protocol.

- The SmartEGM interprets a missing *startDateTime* as Jan 1, 1970 and a missing *endDateTime* as Jan 1, 3030 when the host sends a `download.setScript` command with a Disable Condition field set to **G2S_idle** and no setting for the start and end dates. Previously, the start and end dates were required to be set by the host.

- In previous releases, the SmartEGM Status panel was not updated before the SmartEGM was connected to the remote host. In 1.2 we immediately update the SmartEGM status panel after loading the **smartegm-config.xml** file.

- Some customers may have seen weird behavior from the SmartEGM when their host sent an MSX003 error to the SmartEGM. There were flaws in the logic that handled this error code. We believe that the errors have been removed.

- The SmartHost in the RST 1.1 incorrectly handled the `download.packageStatus` command when sent by the EGM as G2S request. This has been corrected.

- The SmartEGM is now conforms to the G2S specification with regard to handling illegal `communications.getDescriptor` commands. If you send a communications.getDescriptor command with a *deviceId* of zero (0), the SmartEGM now returns the APX003 (Invalid Device ID) error. If you specify a particular device class, the SmartEGM returns a descriptorList for the specified class only.

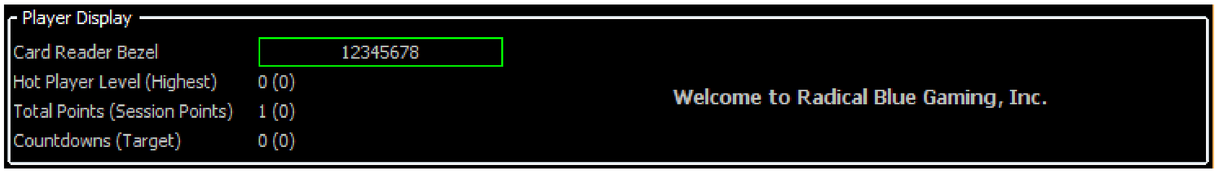
- When the EGM sends the *transferParameters* in the `packageStatus` and `packageLogList` commands it receives from the host, (usually a user name and password) it is now sends the *transferParameters* as asterisks (`***********`).

- A namespace has been added to the mtpCoordination element.
- RST now encrypts the mtp packet if the current key in the mtpCoordination object is not all zeros. If the current key is all zeros, RST sends the packet in clear text.
- SmartEGM Data Model Viewer
    - Packages have been added.
    - Parameters values are updated in real-time.
    - A list of supported events for each device is provided.

About the Player Display

The Player Display shows messages sent from the host to the EGM, including welcome messages, award messages, session messages, and card-out (“goodbye”) messages. The host has the option to use *substitution tokens*, special characters that display as pre-defined information (for example, player name or EGM ID) at the EGM.

If the EGM is not carded on a player-required token, the actual token text displays. For example, if you use the player account number token (%a), you would see “%a” display instead of a player number. Substitution tokens can be used for any message type, and are described in Appendix E of the *G2S Message Protocol* document.



Player Display

Card Reader Bezel	12345678
Hot Player Level (Highest)	0 (0)
Total Points (Session Points)	1 (0)
Countdowns (Target)	0 (0)

Welcome to Radical Blue Gaming, Inc.

To assist testing efforts, player information is displayed in the left-hand corner of the screen.

- **Card Reader Bezel** displays the ID number of the inserted player card. A green border indicates that a player card is inserted; a red border indicates that there is no player card inserted at the EGM.

- **Hot Player Level** displays the current hot player level. **Highest** is the highest hot player level the player has obtained. This field resets at card-in and card-out, and does not require a player card-in for hot player determination to occur.

- **Total Points** displays the total player bonus points (initial point balance + current session points). This field is initially populated by the `playerSessionStartAck` command.

  Note that if you send a `setPointBalance` command after a player session has started, the initial point balance is adjusted to the new value; the number sent is not *added* to the bonus point total.

  Session Points indicates the bonus points accrued for the current session. The current session includes: base point awards, player point awards, generic override points awarded, and points awarded by the host (The individual values are in the player log record). This field is only applicable to carded players.

- **Countdown** displays a player's count until earning a specified number of bonus points. **Target** is the number that the player must reach to receive the bonus point(s). Since you have the option to count down or up, what you can expect to see in this field will change. This field is populated by the `player.setCountdownOverride` command.

  Example

  If you are counting up to 20 with one point earned, you would see:

  ```
  Countdown (Target)  1 (20)
  ```

  If you are counting down from 20 with one point earned, you would see:

  ```
  Countdown (Target)  19 (0)
  ```

New and Modified SmartHost Tiger/XML Verbs

Tiger/XML Verb	Description	Sample Usage*

`tiger:if-vendor`	Allows the script to use different sets of verbs for EGMs from different vendors.	`<tiger:if-vendor tiger:vendor="RBG">`
		`    <tiger:then>`
		`      ...`
		`    </tiger:then>`
		`</tiger:if-vendor>`

`tiger:if-package-available / tiger:then`	Allows the script to perform conditional logic based on the presence or absence of a particular package on the EGM.
	
	To be present, the package must exist in the most recent `package.packageList` command returned from the EGM.
		`<tiger:if-package-available tiger:package-id="RBG_package1">`
		`      <tiger:then>`
		`        <tiger:cabinet-getCabinetStatus>`
		`      </tiger:then>`
		`  </tiger:if-package-available>`
		
		If the package **RBG_package1** exists on the EGM, the host sends the G2S command `cabinet.getCabinetStatus`.

`tiger:download-getPackageList`	Requests the current list of installed packages from an EGM.	
`tiger:download-getDownloadProfile`	Requests the download device profile from an EGM.	
`tiger:download-getDownloadStatus`	Requests the download status from a Tiger/XML script.	
`tiger:download-setDownloadState`	Changes the state of the download device from a Tiger/XML script.	
`tiger:download-addPackage`	Allows a package to be added to an EGM.	

*See the **\scripts\smart-host\rsh-example-download-oo1.xml** Tiger script for a complete example.

`tiger:download-deletePackage`	Allows a package to be deleted from an EGM.	
`tiger:download-getModuleList`	Requests a list of installed modules from an EGM.	
`tiger:download-installPackage`	Installs packages on the EGM.	
`tiger:download-uninstallPackage`	Uninstalls packages from the EGM.	

`tiger:if-device-owner tiger:device-class="G2S_[class]" / tiger: then`	You can now have conditional logic in a Tiger/XML for SmartHost script that works off the owner status for a device. That is, you can do something (or not do something) based on whether the SmartHost is the owner of a given device.	`<tiger:if-device-owner tiger:device-class="G2S_cabinet">`
		`<tiger:then>`
		`<tiger:cabinet-setCabinetState>`
		`</tiger:then>`
		`<tiger:else>`
		`<tiger:cabinet-getCabinetStatus>`
		`</tiger:else>`
		`</tiger:if-device-guest>`
		
		In this example, the `cabinet.setCabinetState` command is sent to the EGM if the SmartHost is the owner of the Cabinet device. If not, the `cabinet.getCabinetStatus` command is sent.

`tiger:if-device-guest tiger:device-class="G2S_[class]" / tiger:then`	You can now have conditional logic in a Tiger/XML for SmartHost script that works off the guest status for a device. In other words, you can do something (or not do something) based on whether the SmartHost is the guest of a given device.	`<tiger:if-device-guest tiger:device-class="G2S_cabinet">`
		`<tiger:then>`
		`<tiger:cabinet-getCabinetStatus>`
		`</tiger:then>`
		`</tiger:if-device-guest>`
		
		In this example, the `cabinet.getCabinetStatus` command is sent to the EGM if the SmartHost is a guest of the Cabinet device.

`core:fail`	This command causes the Tiger/XML script to terminate with an error. You can include the attribute, *core:message*, which is the termination message that is printed in the logger panel.	

`core: exit`	This new verb causes the Tiger/XML script to terminate with success. You can include the optional attribute, *core:message*, which is the termination message that is printed in the logger panel.	
`gamePlay-getGameDenoms`	If you specify the *device-id* as **`-1`**, it will send the G2S command `gamePlay.getGameDenoms` to all known game play devices. This is a shortcut to adding individual `gamePlay-getGameDenoms` verbs.	
`gamePlay.setActiveDenoms`	The `gamePlay.setActiveDenoms` SmartHost verb has been modified with the addition of an *all* attribute that allows you to perform an actio based on whether the SmartHost is the owner of a given device.	`<tiger:gamePlay-setActiveDenoms tiger:device-id="-1"` **`tiger:all`**`="true" /` This example enables all known denominations, on all known game play devices.
`gamePlay.setGamePlayState`	This SmartHost verb now supports a device-id of **`-1`**, which applies the specified command to all known game play devices on an EGM. This allows you to affect all EGM game play devices with a single verb.	
`tiger:Transport.sendMyCommand`	Previously, the `sendRaw` Tiger verb didn't work well if you wanted to send your own commands in the outbound data stream of the tool. This new Tiger verb has an attributes class, deviceId, and command to send. Using this verb, the provided command is inserted into the outbound data stream (`dateTime`, `commandId`, etc. are automatically updated).	<tiger:Transport.sendMyCommand tiger:device-class="G2S_communications" tiger:device-id="-2"> <g2s:keepAlive xmlns:g2s="http://www.gamingstandards.com/g2s/schemas/v1.0.3" /> </tiger:Transport.sendMyCommand>

Version 1.1 [released: March 4, 2008]

We've moved to a new numbering system for 2008, and version 1.1 requires a 2008 license (contact Russ@RadBlue.com if you haven't received your new license). For the foreseeable future, you can expect a new release of our tools around the end of each month. The minor number (x.1) will increment each month, and the major number (1.x) will increment on significant events in the life of the tool.

High-Level Summary

In this newest release, we added the G2S handpay class to the SmartEGM. Significant improvements have been made to the download class, where you can now download, verify, gat, and install a RadBlue package from our website.

New Features

- New versioning convention – RGS has a new versioning convention: [major.minor]. This versioning convention replaces the previous version 1.0.3 build *x*. The last RGS version using the old convention is 1.0.2 build 12.
- Support has been added for the G2S handpay class. See About Handpay for more information.
- Because of the problems with implementing SOAP 1.2, we are dropping back to SOAP 1.1 for the moment, but will pay close attention to the GSA transport committee's progress on this issue.

Improvements

- The SmartEGM Configuration file loader has been enhanced to perform additional semantic error checking on the contents of the file (looking for errors related to the host table, non-existent devices, multiple instances of a single instance class, etc.).

- The SmartEGM now supports filtering based on the attributes in the `communications.getDescriptor` command (for example, a specific class, device, includeOwners, includeConfigs, etc.)

- **download** – The download class support in the SmartEGM was dramatically improved (it was essentially rewritten). Please see the About Download at the end of this document for an extended discussion of the changes related to this class.

- **optionConfig** – SmartEGM now supports `getOptionList` filtering, and has had another round of general tuning. The tool now supports GSA plus third party options. Implementation details for including third party options in the SmartEGM will be provided upon request.

- **gat** – GAT can now verify components and modules (identified in the **smarteg-config.xml** file). With the new download support, downloaded packages, and modules resulting from the installation of a RadBlue package, are also included in the component list, so verifications can also be performed on them.

- **wat** – G2S WAT class support is now complete in the SmartEGM, supporting host and EGM controlled WAT transfers, key exchanges, etc. In the default configuration, WAT device 1 is hostControlled and device 2 is EGM controlled. The choices available on the new WAT Transfers GUI are controlled by the G2S_interfaceMode parameter in the SmartEGM configuration file. See About WAT for an extended discussion.
- EGM Reset button – A new Icon was added to Player Verbs panel that allows the User to reset the SmartEGM's cabinet status to the happy state, in case something gets out of whack.
- **S2S** - The S2S host script now has improved support for the `voucher.authorizeVoucher` command. To make the script more realistic, the `voucher.authorizeVoucher` response is built from the attributes sent by the Edge system through the `voucher.redeemVoucher` request.
- **S2S** – S2S Edge System GUI – A new button is provided to grab the latest URLs from the configuration file.

Corrections

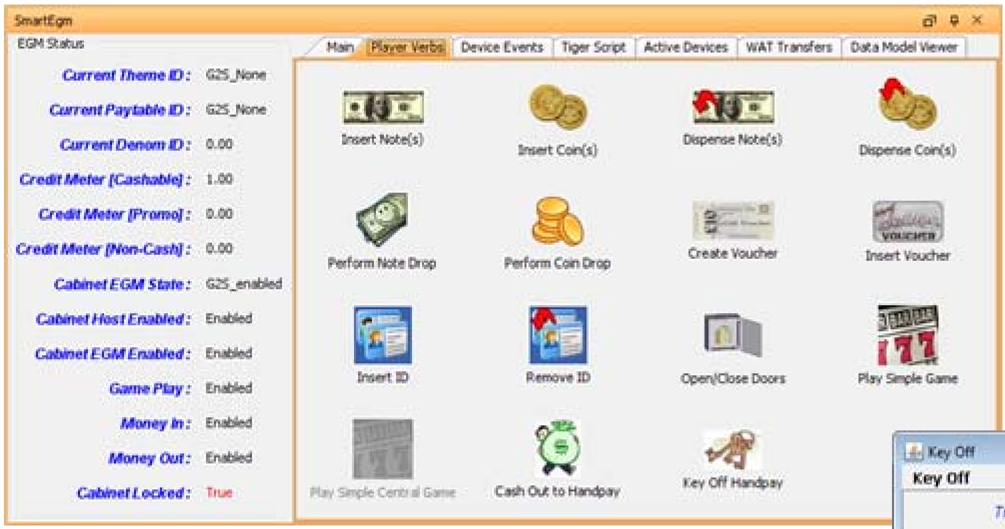
- optionConfig – The SmartEGM now correctly uses the same *transactionId* for each of the commands in an optionConfig sequence.
- The host can now request an option list (`getOptionList`) for a single device. (428)
- In the SmartEGM, the `bonus.cancelBonusAward` command is now handled properly (it used to return a `bonus.error` response). However, if you attempt to cancel a non-existent bonus or a bonus that has already been awarded, a `bonus.error` response is still returned.
- SmartEGM – The bonus log was being accessed by bonusId, which proved problematic. The SmartEGM now uses the transactionId which has a better chance of being unique.
- The `PackageStatus` command sent in response to the `download.addPackage` request now correctly reports the package status. Previously, it reported the package as being available, which was incorrect.
- SmartEGM – now sends a **commsClosing** command when shutting down, even if it is in the Synch state (waiting for the communications device to be host enabled).

About Handpay

- Handpay, voucher and WAT keyoff types are supported. Additionally, handpays resulting from game win can also be paid to the credit meter, and cancel credit handpays can be cancelled.
- Partial handpays are not supported – The entire handpay amount must be keyed off to a single handpay type (cash, voucher, or WAT).
- If a single handpay event results in multiple handpay requests (as is often the case with a cashout to handpay), these can be keyed off individually.
- When keying-off a handpay to WAT, the WAT device with the `cashOutToWat` attribute set to true will be used for the transfer.
- When a handpay is created, the Cabinet goes into an egmLocked state, which is not accessible from G2S. This condition is shown in the Cabinet Locked field in the EGM Status panel. A new GUI button has been added to reset the EGM (if necessary).
- Four new Tiger Verbs have been created (and one has been modified) to support handpay functionality:
    - **Human.playSimpleGame** was modified so you can now specify handpay action (a local handpay will automatically follow) or you can specify REMOTE, in which case the SmartEGM will wait for remote-key-off-timeout milliseconds for a remote keyoff to arrive, after which the win will just go to the credit meter.
    - <tiger:**Human.cashOut** tiger:method="HANDPAY"/> - All the funds in the player's credit meters are sent to a handpay (cancel credit).
    - <tiger:**Human.keyOff** tiger:method="VOUCHER"/> - All outstanding handpay commands will be keyed-off to vouchers (or HANDPAY, CREDIT, or WAT).
    - <tiger:**DataModel.waitForHandpayKeyOff** tiger:timeout = “30000”/> - Script will wait for 30 seconds for a remote keyoff from the host.
    - <tiger:**Human.cancelCancelCreditHandpay**/> - Cancels the outstanding cancelCredit Handpay request.

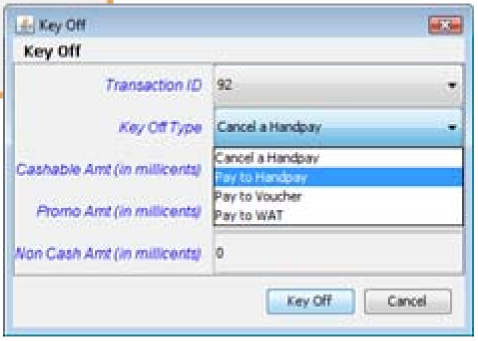
As always, details of all Tiger verbs can be found in the interactive glossary in the Start → Program group for RST.

Using Handpay in RST



EGM Status

Current Theme ID :	G2S_None
Current Paytable ID :	G2S_None
Current Denom ID :	0.00
Credit Meter [Cashable] :	1.00
Credit Meter [Promo] :	0.00
Credit Meter [Non-Cash] :	0.00
Cabinet EGM State :	G2S_enabled
Cabinet Host Enabled :	Enabled
Cabinet EGM Enabled :	Enabled
Game Play :	Enabled
Money In :	Enabled
Money Out :	Enabled
Cabinet Locked :	True



Key Off

Transaction ID	92
Key Off Type	Cancel a Handpay
Cashable Amt (in millicents)	
Promo Amt (in millicents)	
Non Cash Amt (in millicents)	0

Use **Player Verbs** to test various handpay scenarios, such as:

- Create a Handpay: **Insert Note** and then **Cash Out to Handpay**
- Create a Handpay through the **Play a Simple Game** Verb: Select *Win to handpay?* and the type of handpay you want to create.
- Create a Handpay through the **Play a Simple Central Game** Verb: Select *Win to handpay?* and the type of handpay you want to create.
- Keyoff a Handpay: **Key Off Handpay**
- Cancel a Cancel-Credit Handpay: **Key Off Handpay**

About Download

Download has been rewritten in RST to enhance download functionality, add upload capability, and to provide a more complete emulation of the class. Some of the highlights of the rewrite follow:

- *cmdSequence*, used to order the operations inside a given `setscript` command, now has improved handling (commands are ordered, and we check for dupes and skips in the *cmdSequence* values within the `setScript` command).
- The download class now fully supports the `download.addPackage` command. The SmartEGM downloads the specified file through the `addPackage` command. We support two transports:

    1. FTP - if the *transportLocation* attribute starts with **ftp://**, RST uses FTP to fetch the package. The *transportParameters* attribute can specify a username/password combination in the form **username;password**. If the username and password is present and in that form, the SmartEGM will use those given credentials.
    2. HTTP - if the *transportLocation* attribute starts with **http://**, we will use HTTP to fetch the package. The *transportParameters* attribute is currently ignored for HTTP.

    Note that the package file is downloaded to the following directory:

    ```
    ../smart-conf/smart-egm/packages/<EGM-ID>/<package-name>.pkg
    ```

    <EGM-ID> is the ID of the EGM and <package-name> is the name of the package in the `addPackage` command. These files are retained between restarts of both the SmartEGM and the RST. (389)

- The SmartEGM now supports external file references for components and packages.
- Downloaded packages are automatically added to the SmartEGM configuration file, so if you restart the tool using the "-updated" version of the configuration file, the new files are persisted.
- Downloaded packages and installed modules can be seen and verified through `gat.componentsList`.

The SmartEGM configuration file has been modified to help control FTP and HTTP package transfers with the addition of four new attributes:

- *http-transfer-timeout* – The timeout (in milliseconds) to use when using HTTP to transfer a package.
- *http-transfer-max-retries* – The maximum number of retry attempts when using HTTP to transfer a package.
- *ftp-transfer-timeout* – The timeout (in milliseconds) to use when using FTP to transfer a package.
- *ftp-transfer-max-retries* – The maximum number of retry attempts when using FTP to transfer a package.

These transfers now have connection timeout values and retry control. Read the SmartEGM Config File XSD for more info on these attributes.

Using Download Packages

A fully functional package system has been added to RST.

When the SmartEGM gets a .zip file, it looks in the root directory of the .zip for a file called "package.xml."

Once found, SmartEGM uses the new package definition XSD file to validate the contents. If the package content passes validation, the content is used to define the package information: in the XML file, you define the package, module and storage information. Each module points to a physical file in the .zip file that represents the module file in a real EGM. It is the same information as in the G2S XSD, but bundled into a .zip file.

Once the package is loaded into the SmartEGM, you can test various Download scenarios, such as install a module, uninstall a module, and read package contents.

Creating a RadBlue format package

Finally, you can create a *new* package out of existing modules by creating a new .zip file with the **package.xml** file, and then transferring it to an FTP site.

When you add a package on the SmartEGM, or install the package to create modules, you can verify the installation by looking on the disk in the following areas:

```
../smart-conf/smart-egm/${EGM-ID}/modules/
```

```
../smart-conf/smart-egm/${EGM-ID}/packages/
```

A module is defined by a single file, and each module in a package has its own directory. The .zip file may contain more than one file per module. The entire contents of the .zip file are unpacked.

Once you have downloaded a package or installed a module, you can view those items by using the `gat.getComponentList` command. Packages have a *componentType* of **G2S_package**, and modules have a *componentType* of **G2S_module**.

The Package Definition XSD file defines the GAT authentication algorithms supported by the module. This information is also returned in the component list. Once you know a packageId/moduleId and a supported GAT authentication algorithm, you can execute a `gat.doVerification` command.

Sample Package Definition XSD File

An XSD file that defines the package.xml file we use for the package directory (defining the contents of a RadBlue SmartEGM package file) is part of the SmartEGM and shipped as part of the RST installer. You can find it in the Start menu. The SmartEGM uses the package.xml file to determine what modules are in the package, and what algorithms can be run against each of the modules.

Our sample package is located here:
http://www.radblue.com/downloads/smartegm/packages/package-1.zip

About WAT

The G2S WAT class support is now complete in the SmartEGM, supporting host and EGM controlled WAT transfers, key exchanges, etc. In the default configuration, WAT device 1 is host controlled and device 2 is EGM controlled. The choices available on the new WAT Transfers GUI are controlled by the G2S_interfaceMode parameter in the SmartEGM configuration file.

G2S_hostControlled transfers – If the WAT device is configured with a hostControlled interface, the following actions are available:

1. Get Key Pair – sends a getKeyPair request to the host.
2. Insert ID – Allows you to simulate an ID being inserted (which you can also do via the Player Verb GUI). Once you have inserted an ID, this icon is no longer active (With RGS, use ID number **`12345678`**).
3. Cash Out – If the WAT device you are working with is the device identified for WAT cashouts, the CashOut button will also be active (*watStatus.cashOutToWat* attribute = “true”).
4. Remove Id – If an ID has been inserted, this icon is active and will cause the ID to be removed.

G2S EGM Controlled transfers – If the WAT device is configured with an egmControlled interface, the following actions are available:

1. Get Key Pair – sends a `getKeyPair` request to the host
2. Insert ID – Allows you to simulate an ID being inserted (which you can also do through Player Verbs). Once you have inserted an ID, this icon is no longer active. (With RGS, use ID number **`12345678`**)

Once a valid ID has been inserted, the following icons become active:

1. Get Accounts – Sends a request to the host for a list of player accounts. The details of each account are shown on the tab at the bottom of the control.
2. Get Balance – Requests the balance for the account(s) returned through the **getAccounts** inquiry.
3. Transfer Funds – Allows you to initiate a transfer in either direction between the EGM and the host.
4. Cash Out – If the WAT device you are working with is the device identified for WAT cashouts, the Cash Out button will also be active (*watStatus.cashOutToWat* attribute = “true”)
5. Remove ID – If an ID has been inserted, this icon is active and, if selected, causes the ID to be removed.