Game Authentication Terminal Program (GAT3)

Requirements Document

Doc. ID: gsa-p0049.001.01

Gaming Standards Association
Technical Committees

Document Released: August 22, 2005

GDS Committee, Game Authentication Terminal Program (GAT3), Requirements Document gsa-p0049.001.01

Released on August 22, 2005, by Gaming Standards Association (GSA).

Patents and Intellectual Property

The user's attention is called to the possibility that implementation of the Gaming Standards Association (GSA) standard or specification contained herein may require the use of inventions/ technologies covered by patents or other intellectual rights held by third parties. By publication of this GSA standard or specification the GSA makes no representation or warranty that the implementation of the standard or specification will not infringe on any third party rights. The GSA takes no position with respect to any claim that has been or may be asserted by any third party regarding intellectual property rights or [on] the validity of any such rights related to any such claims, or the extent to which a license to use any such rights may or may not be available.

Trademarks and Copyright

GSA Contact Information

GSA – Gaming Standards Association
48377 Fremont Blvd., Suite 117
Fremont, CA 94538

Phone: +1(510) 492-4063

Fax: +1(510) 492-4001

E-mail: sec@gamingstandards.com

WWW: http://www.gamingstandards.com

Table of Contents

About This Document

Bally Gaming & Systems worked together with the New Jersey Division of Gaming Enforcement (NJ-DGE) to develop the Game Authentication Terminal (GAT) application, used in the field by regulatory field agents to identify casino games, and the game software or firmware component versions or revision codes, in the field.

This document defines the requirements and design of the GAT3 application, which is an enhanced, refined version of the original. It also describes the changes that were made between the original GAT and the new GAT3, and why those changes were made.

Bally Gaming & Systems designed and created GAT3, and has made it available to GSA to modify and incorporate its protocol into GSA standards, such as the BOB Message Protocol and GDS device protocols.

GSA

The Gaming Standards Association(GSA) is an international trade association representing gaming manufacturers, suppliers, operators and regulators. We facilitate the identification, definition, development, promotion, and implementation of open standards to enable innovation, education, and communication for the benefit of the entire industry.

For more details about GSA, visit the Web site: http://www.gamingstandards.com.

Related Documents

The following documents are referenced throughout this design document.

Ref	Title and Source
NV1	Nevada "Proposed Amendments to Technical Standards for Gaming Devices section 1.080 2(d)
NJ1	New Jersey TAB01-001
GSA-SVC	GSA SVC Serial Protocol v1.00, document ID gsa-p0011.001.00. (standard adopted July 19, 2000)
NIST-FIPS-140-1	National Institute of Standards and Technology (NIST), Federal Information Processing Standard (FIPS) FIPS 140-1: Security Requirements for Cryptographic Modules, January 4, 1994

Document Conventions

`Courier New text indicates directory paths, file names, code or pseudo-code.`

Blue text most often indicates active cross-references to other sections within the document, but may also indicate active hyperlinks to Web pages.

Term Definitions

DSN	Data Source Name
EGM	Electronic gaming machine
GSA	Gaming Standards Association
GAT	Game Authentication Terminal
NJ-DGE	New Jersey Division of Gaming Enforcement
ODBC	Open Database Connectivity
SVC	Serial Verification Communication protocol
XML	Extensible Markup Language

Revision History

The following table lists the changes made this document.

Revisions

Version	Date	Changes
1.0	8-22-2005	No version change. Corrected contact phone number and removed “GSA Confidential” from page footers.
1.0	4-22-2005	Converted to GSA format and minor edits for clarity.
0.4	10-25-2002	Bob Dubner - changes to XML default “components” response.
0.3	10-10-2002	Bob Dubner
0.2	10-10-2002	Edited by Jim Morrow
0.1	10-07-2002	Released to Jim Morrow for comment.

1 Introduction

New Jersey and Nevada gaming regulations are being modified to include a requirement for a means to validate all program components on demand via a communication port (see document references NV1 and NJ1), the purpose of which is to allow regulatory field agents to efficiently verify games on a casino floor.

Starting sometime in the latter half of the year 2000, The New Jersey Division of Gaming Enforcement(NJ-DGE) and Bally Gaming & Systems started working together on a system that came to be known as the Game Authentication Terminal (GAT).

The primary purpose of GAT is to facilitate the identification of casino games in the field, and in particular, the versions or revision codes of various components, usually software or firmware, included in those games.

GAT was deployed early in 2001, and has been in use in New Jersey since that time.

The system has been successful enough that the NJ-DGE wishes to expand its use to equipment from other manufacturers. Unfortunately, the existing GAT program lacks flexibility and generality; in its existing form, it is designed rather closely around Bally's requirements. Even more unfortunately, changes or extensions to the GAT functionality require changing the GAT source code.

At one time, we thought the solution to that problem would be to simply make the source code openly available to anybody who wanted to use it. That turns out to be unworkable. If the industry ends up with different versions of GAT from each manufacturer, and then different versions of GAT from single manufacturers as the capabilities and needs of their gaming systems change, the regulators are back to square one—instead of a single program that can identify multiple machines from multiple manufacturers, they would have to select the appropriate program for any given game.

Instead, we have designed and created GAT3. GAT3 incorporates the capabilities of the previous GAT system, so that it can talk to older GAT-capable Bally games properly. But it also greatly expands the flexibility of the system. GAT3 has the functionality to do most of what is needed in general; it also includes hooks for doing just about anything without any concomitant need to change the GAT3 code.

In the following sections we describe how the older GAT system works. Following that, we detail the functionality of GAT3.

2 The Legacy GAT System

2.1 Physical Interconnection Layer

The GAT program is a Microsoft® Windows® application, typically running on a notebook computer. It talks to a GAT-capable electronic gaming machine (EGM) by means of a 3-wire RS-232 serial connection. (The three wires are GROUND, TRANSMIT, and RECEIVE.)

In Bally equipment, the serial port used for GAT communications is behind a locked access door. Because the door is open, the EGM can be placed into a maintenance mode which includes the GAT "slave" capabilities.

2.2 Data Transport Layer

GAT talks to the "slave" EGM over the serial line using the GSA SVC Serial Protocol (see document reference GSA-SVC). A copy of the GSA *SVC Serial Protocol* document should be included in the same package as this document. Failing that, request a copy from GSA (contact information available on the Copyright page). The SVC Protocol was proposed by GSA for exactly this purpose—that is, to communicate with an EGM over a serial line for the express purpose of identification and authentication.

The implementation of SVC used by GAT complies with the published standard in all ways except one: SVC has rather stringent inter-character response time requirements. If there is more than a five millisecond delay between two of the 9600 baud characters, that's supposed to be flagged as an error.

GAT relaxes that specification to 500 milliseconds before giving up on an expected incoming character, and does not report it to the operator. The philosophy behind that design decision: GAT is a completely interactive program; if the operator does not see an expected response, the operator will simply click the appropriate button again.

The SVC Protocol is straightforward. It allows the SVC "master" (in this case the GAT program) to make four different requests of the SVC "slave", as follows:

- What is the status of the EGM?
- Initiate an authentication calculation.
- What is the status of the most recent authentication calculation?
- *Return the next block of the authentication calculation response.*

2.3 Legacy GAT Functionality

2.3.1 What Legacy GAT Sends

GAT uses all four of the SVC commands. The one of most interest is the *Initiate Authentication Calculation Query*.

As can be seen in the SVC specification, that query must be sent with a single-byte Authentication Level code. In the legacy GAT program, that authentication level is set to one. The optional Authentication Seed is not used at all in legacy GAT.

2.3.2 What Legacy GAT Expects Back

Legacy GAT (and GAT3, if the legacy **Auth. Level 1** button is pressed) expects a response file back from the EGM.

The contents of that response file must never change!

That response file must be *exactly* the same from every game of the same type whenever an Authentication Level 1 query response is calculated.

2.3.3 What Legacy GAT Does With The Response

When legacy GAT receives a response from an EGM, it first runs the entire response through the SHA-1 Secure Hash Algorithm (see document reference NIST-FIPS-140-1). This creates a 20-byte digest of the original file. That 20-byte binary number is converted to a 40-character hexadecimal string. That string is known as the "hash".

What GAT does with that hash depends on whether it is running in Laboratory or Field mode, as described in the following sections.

2.3.3.1 GAT Laboratory Mode

GAT operates in two modes, which are known as Laboratory and Field. In both cases it is exactly the same program, running the same way. But the Laboratory computer also has the `GAT.MDB` Microsoft® Access® database file installed in it. That database file is set up as an ODBC System DSN with the name GATDATA.

The legacy GATDATA database has a very simple hierarchical structure. There can be a number of `Manufacturer`s. Each manufacturer can have associated with it any number of `Game`s. Each game can have associated with it any number of 40-character `Hash`es.

After hashing a response from an EGM, GAT looks to see if the GATDATA ODBC source exists. If it does, GAT decides that it is running in Laboratory mode. It looks for the digest string in the database. If found, GAT reports the manufacturer and game associated with that hash value.

If the hash value is not already in the `GAT.MDB` database, it offers to let the operator associate the new hash value with a game already in the database. (Note: the `Manufacturer` and `Game` must be entered manually *before* the association can be made.)

The `GAT.MDB` database has a built-in function that dumps the entire database in a simple hierarchical structure in a file named `GATDATA.TXT`.

2.3.3.2 GAT Field Mode

The field agents' notebook computers run the same GAT program. But instead of the `GAT.MDB` database, they instead get loaded up with copies of the `GATDATA.TXT` file, which, obviously, must be updated from time to time.

When legacy GAT receives a response file from the EGM, it hashes it. It scans the `GATDATA.TXT` file looking for a match. If it finds a match, it reports the `Manufacturer` and `Game` to the operator.

If it does not find a match, it says so.

And that, oddly enough, is that. Right there is the whole purpose of the GAT system, and the GAT terminal, and the SVC protocol, and all that stuff. You plug the GAT terminal into a game, press the button, and it tells you what game you are talking to.

The point, of course, is that the `GAT.MDB` database is updated periodically, and those updates are reflected in the `GATDATA.TXT` file downloaded to the field agent's machines, including useful information such as, "THIS VERSION IS OBSOLETE AND THE MACHINE MUST BE UPGRADED."

3 Legacy GAT Works, Why Change It?

We have to change legacy GAT because its level of specificity is too coarse. In retrospect, it was a mistake to hash the entire response file.

As emphasized earlier, in response to an Authentication Level 1 Query, the returned response has to be exactly the same for an entire class of machines. Useful information such as date stamps, manufacturing dates, and serial numbers *must* be omitted from the Level 1 response, because the information changes from machine to machine, and from time to time, and thus would change the hash value.

This problem became acutely obvious in mid-2002, after GAT had been in use in New Jersey for about a year. Below is an actual response from a Bally game to a GAT Level 1 request:

Figure 3.1 Legacy GAT: Response to a GAT Level 1 Request

```
[Authentication File Data]

Authentication Level : 1
Authentication Seed  : N/A
Manufacturer         : Bally Gaming & Systems, Inc.
Model                : EVO V8700

[Authenticated Item Data]

BIOS+:
 Checksum
 V7S0100BIOSP-01 : 20c9

CD-ROM:
 SHA-1/DSA
 V7GC7GC2SL00-31 : ca 4f 44 5a 5d 79 ed b6 58 4d ba ed 21 48 e4 d4 62
 24 73 4f dc 72 1d 4a 82 a8 53 f9 a7 d8 97 7d 21 18 04 cf 95 6a 54 6d

MPU EPROMs:
 Checksums
 V7M1E02C7204-21 : F68A
 V7M4E02C7204-21 : BD13
 V7PY43768702-00 : E659
```

As has been repeatedly emphasized, the hash of the entire response becomes the data that links back to the game in the database.

The problem surfaced when Bally Engineering made a change to one of the MPU EPROMs. That caused the part number of the EPROM, and its checksum, to change. This in turn caused "GAME NOT FOUND" responses to pop up when GAT was used on those updated games in the field.

It turns out that NJ-DGE had no interest in tracking EPROM changes with GAT. They have other methods, in place for years, for tracking PROM changes; their main interest in GAT was that it gave them a way of tracking changes to the CD-ROMs inside the new games that used them.

The solution: (Your author also wrote the GAT program; please imagine him cringing as he writes this next line.) The internal code of GAT was altered. If a line starting with the word "Manufacturer" also contains the word "Bally", and if a line containing the word "Model" contains the word "V8700", then *only* the three lines following the first line starting with the string "CD-ROM:" are included in the hash.

This is, of course, horrid and unacceptable. In order to be useful in the future, the functionality of GAT *must* be expanded to be more flexible.

With that understanding of legacy GAT's limitations in mind, we look now at GAT3, which addresses those problems.

4 GAT3 Requirements

- Facilitates the identification of casino games and their components in the field.
- Meets the requirements outlined in document reference NV1 and
- Meets the requirements outlined in document reference NJ1.
- Produces a file output in response to an authentication query.
- Provides a database that can be used by lab and field regulatory engineers to verify game content. The database or its equivalent must be capable of running on laptop computers running Microsoft® Windows® 95/98/2000/XP. In other words, it must be able to run on old laptop PCs.
- The GAT program should not require expensive programs or costly upgrades to a field agent's laptop PC. In other words, neither the regulators nor Bally nor other game manufacturers want to purchase new software or hardware to make GAT work.
- The regulatory lab computer can be expected to require a certain level of operating system and program. In other words, the lab PC can be expected to run Microsoft® Windows® 2000 at least and have Microsoft® Access® 2000 database program installed.
- It is called **GAT3**

  At Bally, there is some minor confusion about whether the legacy system in use in New Jersey should properly be called GAT or GAT2. Calling the expanded system GAT3 does an end run around that nomenclature problem.
- GAT3 provides for expanded component tracking

  Legacy GAT, as has been said, is too coarse. There is one, and only one, possible response per game entity. Games, however, are typically comprised of a number of trackable entities—several different EPROMs and CD-ROMs and so forth. GAT3 provides a mechanism whereby a game can report any number of trackable components. In other words, GAT3 allows finer incremental tracking of the software components in an EGM.
- GAT3 provides for expanded command capability

  After you have a notebook computer attached to a game, it seems a shame to limit the possibilities to a simple, "What Are You?" question. GAT3 provides a mechanism whereby the game provides GAT3 with a list of possibilities. The user then indicates which of those possible functions are to be performed.

  With this structure it is not generally necessary for GAT3 to know what the capabilities of any particular game are—it serves as a conduit between the operator and the game in terms of executing commands, even though it does not know what those commands are or what they mean.
- GAT3 provides for expanded customized execution capability.

  One of the very real limitations of legacy GAT involves its limited capability to respond to responses from a game; if the author of GAT did not think of it, then GAT cannot do it.

  GAT3 provides a mechanism whereby the response from a game can itself trigger an additional program to process that response. In this fashion, GAT3 becomes a universal

data transfer conduit between a manufacturer's game and additional software provided by that manufacturer on the GAT3 notebook computer.

- GAT3 allows for the inclusion of seed values in responses.

  Some authentication techniques may require the inclusion of a passphrase or secret known to the user. GAT3 shall allow for this by an INI file.

- GAT3 allows for the regulatory authority to obsolete components and indicate such in the database and for field agents.

5 Implementation of GAT3

5.1 Basic philosophies

5.1.1 Co-opting by GAT3 of SVC Authentication Level 186

As detailed in section 4, GAT3 needs to do many things that the SVC protocol does not provide for directly. Thus, GAT3 reserves a single Authentication Level code, decimal 186, for its own use.

Because it is not believed that the SVC protocol is in use by any other organization at this time, and because of the relative weirdness of decimal 186, it is hoped that there will never be any kind of general conflict caused by that value being used by any other manufacturer that might use the SVC protocol for something other than GAT3 communications.

The number 186 was chosen because of its hexadecimal representation, BA, which we here at Bally thought amusing.

5.1.2 Use of XML in GAT3

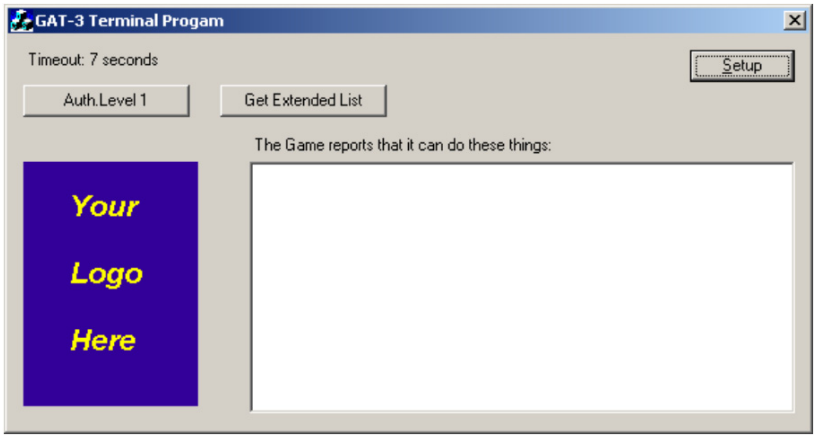
In general, when GAT3 sends a request to a game, it looks at the file that comes back. If that file is an XML file, GAT3 will attempt to process that file according to rules that will be discussed in sections that follow. If a file is not in XML format, GAT3 will simply store it and ignore it.

GAT3 is intended for use local to the game. That is, a person will be standing right there, connected to the game when GAT3 is used. However, the designers recognize the advantages of passing GAT3 messages over a network back to a host. By using XML in GAT3, the designers hope to make this dual use of GAT3 messages: local and to a server, less painful for the industry.

5.2 Starting up GAT3 – IDLE State

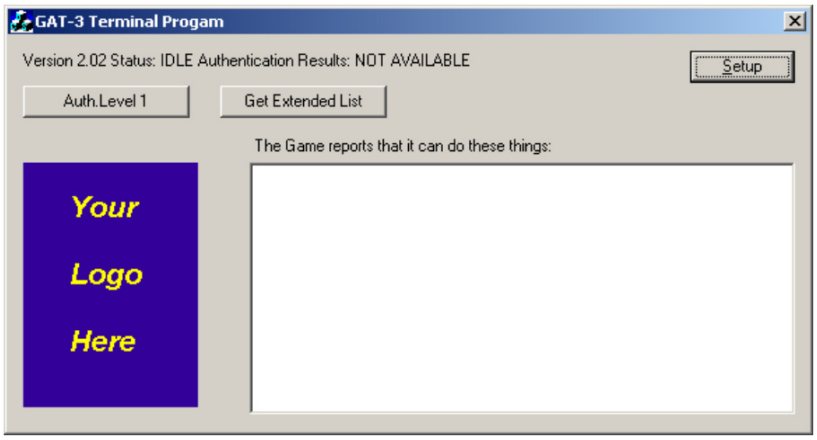
When you start the GAT3 application, a screen opens similar to the following:

Figure 5.1 GAT3 Start Up



The status message, "Timeout 7 seconds," indicate that the program has been running for seven seconds since it last had contact with a GAT-capable EGM. SVC *Status Query* commands are transmitted several times each second; if the GAT3 terminal is connected to a GAT-capable EGM, the screen appears similar to the following.

Figure 5.2 GAT3 Connected to GAT-capable EGM



The status information indicates a working connection between GAT3 and the EGM.

The various elements of this screen are described in Table 5.1.

Table 5.1 Description of Buttons in Main Screen

Button	Description
Setup	Click this button to open a screen that allows for selection of the serial port to use. It also allows the operator to specify where GAT3 should store the files that are created from the information returned from the EGM in response to *Initiate Authentication Calculation Query* commands. The logo area is filled with a 125 by 175 pixel LOGO.BMP file, which is co-located with the GAT executable. For the New Jersey implementation, the logo of the NJ-DGE is placed there; it is anticipated that various jurisdictions might enjoy doing the same.
Auth. Level 1	Click this legacy GAT button to send an SVC *Initiate Authentication Calculation Query* with the Authentication Level set to one. The response that is returned is processed as described in the Legacy GAT section (see section 2).
Get Extended List	Click this button to access GAT3 capabilities. See section 5.4

5.3 Special Commands that GAT3 Understands

5.3.1 Change Baud Rate

GAT3 understands the `Change Baud Rate` command; after sending this command and receiving a response from the game, it will close and re-open its serial port using the new baud rate.

5.3.2 Get File

GAT3 also understands the `Get File` command. This command may have multiple parameters, but it always uses the first parameter as a file name. The response file will be saved in the response directory under that name. (Any path information will be ignored.)

5.3.3 Get Special Functions

There is always one additional command that GAT3 understands and will issue to a game: The `Get Special Functions` command, which GAT3 issues to the game to get the list of functions the EGM supports, as in the example shown in Figure 5.3.

5.3.4 %% Replacement

Any command can have embedded in it strings of the form seen above, %%<something>%%. When GAT3 sends a command containing one or more of those strings, GAT3 looks for a `SEEDS.INI` in the same folder as the `GAT.EXE` executable. It checks the `Seeds` section for a key matching the <something> string inside the doubled percent signs.

If found, the value of that key replaces the entire %%<something>%% string. If the key is missing from the `SEEDS.INI` file, a value of `"(none)"` is used.

This provides a mechanism whereby individual field agent laptops can be customized with individual strings. Because there is only one `Seeds` section in the initialization file that must be shared by all manufacturers, each manufacturer should prefix their seed codes with something to indicate a manufacturer-specific code in order to avoid the collisions that would occur if everybody chose to use the same word, such as "Seed". For example, ABC Example Corp. might prefix their seed codes with "ABC".

5.3.5 Expansion Commands

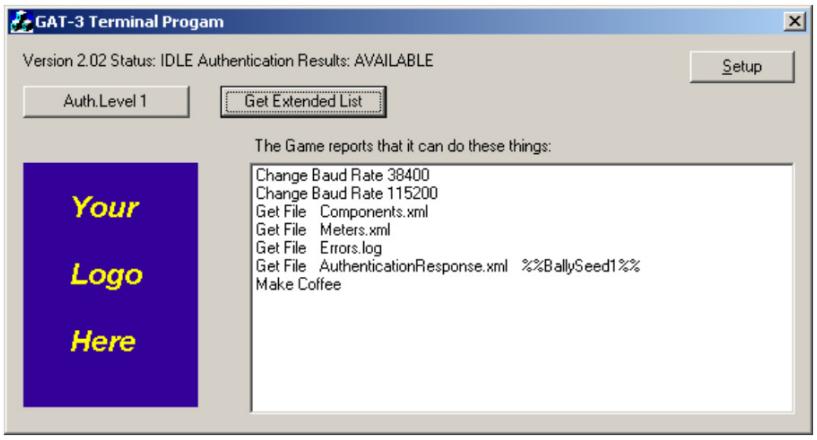
The `Make Coffee` command is one that is unknown to GAT3. If selected, GAT3 will simply send it back to the EGM. GAT3 expects a response file, which will be given a default name and processed as if it were any other response file. Other than that GAT3 will take no special action.

5.4 Getting Specific: How the Get Extended List Button Works

5.4.1 The Interface

Click the **Get Extended List** button and GAT3 issues the `Get Special Functions` command to the EGM. The EGM responds with a list of supported commands and these are presented in the GAT3 screen. Figure 5.3 shows an example list returned.

Figure 5.3 Example List of EGM-supported Functions



Double-click a command to select it and GAT3 sends the selected command to the EGM (as defined in section 5.4.2). As each command is sent, GAT3 examines it to see if it is one of the commands for which special action can be taken (see section 5.3). In the example in Figure 5.3, GAT3 would take special action with the `Change Baud Rate` and `Get File` commands.

Otherwise, GAT3 simply passes the command back to the game, and takes no special action. It does save the response file to the *Initiate Authentication Response Query* with a default name, and processes it as it would any file.

5.4.2 What GAT3 Sends Out

When the **Get Extended List** button is clicked, GAT3 puts together an *Initiate Authentication Calculation Query* command with an authentication level of $186_{10}/BA_{16}$. When that command is issued, additional special GAT3 information goes into the Authentication Seed field.

The very first byte of the Authentication Seed field is reserved as a sub-command byte. At this point in time, only zero has been assigned, which indicates that the remainder of the Authentication Seed field is to be interpreted as a command.

In this particular case, the 27 hexadecimal bytes of the command are:

04	SVC *Initiate Authentication Calculation Query*
1B	Length of 27 bytes for the whole packet
BA	This is a co-opted "GAT3" command
00	Process remainder of seed as a "special command"
47 65 74 20	"Get"
53 70 65 63 69 61 6C 20	"Special"
46 75 6E 63 74 69 6F 6E 73	"Functions"
2B 54	CRC16 Checksum

Note that the string is not terminated with a zero character; the end of the string is implicitly defined by the length of the packet.

C/C++ source code for the CRC16 checksum specified by the SVC Protocol is included in this package.

5.4.3 What Response is Returned

The response to the `Get Special Functions` command is an XML file. The format of this file is of especial interest to game developers, because GAT3 will only be able to process the returned result if it matches this form precisely.

The response file that generated the list box sample seen in Figure 5.3 is shown below:

Figure 5.4 XML Response File Example

```
<?xml version="1.0"?>
<SpecialFunctions GatExec="default">
   <Function>
      <Feature>Change Baud Rate</Feature>
      <Parameter>38400</Parameter>
   </Function>
   <Function>
      <Feature>Change Baud Rate</Feature>
      <Parameter>115200</Parameter>
   </Function>
   <Function>
      <Feature>Get File</Feature>
      <Parameter>Components.xml</Parameter>
   </Function>
   <Function>
      <Feature>Get File</Feature>
      <Parameter>Meters.xml</Parameter>
   </Function>
   <Function>
      <Feature>Get File</Feature>
      <Parameter>Errors.log</Parameter>
   </Function>
   <Function>
      <Feature>Get File</Feature>
      <Parameter>AuthenticationResponse.xml</Parameter>
      <Parameter>%%BallySeed1%%</Parameter>
   </Function>
   <Function>
      <Feature>Make Coffee</Feature>
   </Function>
</SpecialFunctions>
```

The features in *italics* are the ones that game developers will replace with their own specific commands. The rest of the file must be regarded as a template.

Note that every `Function` must have a single `Feature`, and that a function can have any number of `Parameters`.

5.5 Returning a Function to the Game

When a `Function` is selected for transmittal back to the game, GAT3 builds an SVC *Initiate Authentication Response Query* with the authentication level set to 0xBA.

The first byte of the Authentication Seed is set to zero.

The `Feature` text of the `Function` is then appended.

The various `Parameters` will follow, in the order they appear in the XML file. ASCII <tab> characters will be used as separators; the command line ends, however, with the last character of the final `Parameter`.

Thus, the first `Function` in Figure 5.4 would be returned to the game in the Authentication Seed field as:

"Change Baud Rate<tab>38400".

The final `Function` up above would be returned to the game as:

"Make Coffee"

5.6 How GAT3 Processes EGM Response Files

Every file that comes back from the EGM in response to a GAT3 command is examined by GAT3. There are five possible classes of responses described in the following sections.

5.6.1 Response File is not an XML File

In this case, GAT3 does nothing except save the file in the response folder.

5.6.2 Response is an XML File, but There is no GatExec= Attribute

GAT3 does nothing except save the file in the response folder.

5.6.3 Special Functions Response

If the response is an XML file, and if the root element is `SpecialFunctions` with the attribute `GatExec= "default"`, it is parsed as a Special Functions file; each `Function` element is placed into the GAT3 list box as described above.

5.6.4 Default component processing response

If the response is an XML file, and if the root element is `Components` with the attribute `GatExec= "default"`, it is parsed as a default-style Component file.

Here, in Figure 5.5, is a default Component XML response. It is based on the legacy GAT response shown earlier in Figure 3.1 (the one that demonstrated the flaws in legacy GAT). It can be used as a template for GAT3-compliant games.

Figure 5.5 Default Component XML Response

```
<Components GatExec="default">
   <Game>
      <Name>EVO V8700 "Hot Ticket"</Name>
      <Manufacturer>Bally Gaming &amp;amp; Systems, Inc.</Manufacturer>
      <Component>
         <Name>BIOS+: V7S0100BIOSP-01</Name>
         <Checksum>20c9</Checksum>
      </Component>
      <Component>
         <Name>CD-ROM: V7GC7GC2SL00-31</Name>
         <Checksum>ca4f445a5d79edb6584dbaed2148e4d462...</Checksum>
      </Component>
      <Component>
         <Name>MPU EPROM: V7M1E02C7204-21</Name>
         <Checksum>F68A</Checksum>
      </Component>
      <Component>
         <Name>MPU EPROM: V7M4E02C7204-21</Name>
         <Checksum>BD13</Checksum>
      </Component>
      <Component>
         <Name>MPU EPROM: V7PY43768702-00</Name>
         <Checksum>E659</Checksum>
      </Component>
   </Game>
</Components>
```

As before, the information shown in *italic* characters must be replaced on a game-by-game basis. Although the `Components` for only one `Game` are shown here, the `Components` root element can have multiple `Game` child elements.

When GAT3 processes this file, it calculates a 40-character digest for each component. The digest is based on the `Name` and `Manufacturer` of the `Game`, and `Name` and `Checksum` of the `Component`, for each component.

In Laboratory mode (see section 2.3.3.1), this information is stored in the `GAT.MDB` database, which is expanded for operation with GAT3. Each `Game`, in addition to the legacy `Hashes` entries, also can have many `Components`. The `Name` and `Checksum` of the `Component`, along with the calculated digest, are stored for each component.

In Field mode (see section 2.3.3.2), the calculated digest is used to index the `Manufacturer`, `Game`, and `Component`.

In this fashion, the program will obviously come up blank on a component that is not already in the database. In addition, later on the lab can flag a component in the database as, for example, obsolete, viz., "MPU EPROM: V7PY43768702-00 – OBSOLETE!!", and that's what the field agent will see when GAT3 presents them a list of what components are reported to be in any particular game.

5.6.5 Customized processing response

If the response is an XML file, and if the root element has the attribute `GatExec= "path\program.exe"`, then the mechanism that allows completely general custom processing is brought into play.

This is best shown, perhaps, by an example:

Suppose that the GAT3 executable is located at

```
C:\Program Files\Gat\Gat.exe
```

and is set up to store incoming data files in

```
C:\GatData\
```

Consider a game that presents the command

```
Get File meters.xml
```

Suppose further the `meters.xml` file starts off with a root element that has the attribute

```
GatExec= "Bally\ProcessMeters.exe"
```

The result of those conditions is this: GAT3 will save the incoming `meters.xml` file, and then spawn a new process with this command line

```
"C:\Program Files\Gat\Bally\ProcessMeters.exe" "C:\GatData\meters.xml"
```

In this fashion, any manufacturer can work with any regulator or casino to install custom programs to handle custom data, with the data itself indicating what program is supposed to process it. In this way, GAT3 simply becomes a common means of collecting that data and firing off those programs; but, aside from that, has no involvement.

6 Egm Emulator

The GAT3 distribution package includes an EGM Emulator program, `egmemulator.exe`. When the GAT3 setup routine is used to specify that GAT3 communicate on COM9, the program attempts to invoke an `egmemulator.exe` program in the same folder as the `gat.exe` executable. The EGM Emulator then creates a Windows "named pipe", to which GAT3 then connects instead of using a serial port.

The EGM Emulator program behaves like a game. In response to the Special Functions request, it produces a menu much like the one shown in this document.

It is quite flexible. The list of files it creates is not hardwired. Instead, the EGM Emulator looks for a folder named `\GatLogs\` on the same disk drive as the `egmemulator.exe` executable. The list of files it creates is made up of all of the files in that folder that have `.XML` or `.LOG` extensions.

In this fashion, you can create your own `.XML` files and place them in the `\GatLogs\` folder. GAT3 will process those files exactly as if they had been downloaded from a real game. It is anticipated that the GAT3 + EGM Emulator combination will prove very useful both to GAT3 users and game manufacturers for training, familiarization, experimentation, and development.

7 Conclusion

We have tried to show how the existing legacy GAT program requires expansion.

We have described GAT3, which provides for greater flexibility in downloading files. It provides for more options in the default processing of various components of single games, and for complete customized processing of data.

It is hoped that default and custom options will go a long way towards making this program useful to all manufacturers and casinos, and that the features are documented well enough so that they actually can be used.

END OF DOCUMENT

gsa-p0049.001.01
Released: August 22, 2005