Requirements for the Game Authentication Terminal Program (GAT3)

Robert Dubner
201-664-6434
rdubner@dubner.com

James Morrow
Bally Gaming and Systems
6601 S Bermuda Rd
Las Vegas, NV 89119-3605
morrowj@ally.com
702-837-4080

DATE	REV.	DESCRIPTION
2002-Oct-07	0.1	Released to Jim Morrow for comment
October 10, 2002	0.2	Edited by Jim Morrow
October 10, 2002	0.3	Bob Dubner
October 25, 2002	0.4	Bob Dubner – changes to XML default “components” response

TABLE OF CONTENTS

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. Reference Nevada "Proposed Amendments to Technical Standards for Gaming Devices section 1.080 2(d) and New Jersey TAB01-001. The purpose of such 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 and Bally Gaming & Systems started working together on a system that came to be known as the "Game Authentication Terminal," or "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 then.

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, then the regulators are back to square one—instead of a single program that can identify multiple machines from multiple manufacturers, they'd have to select the appropriate program for any given game.

So, 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 next sections we describe how the older GAT system works. Following that, we'll detail the functionality of GAT3.

2. The Legacy GAT System

2.1. Physical Interconnection Layer

The GAT program is a 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.

Updated: October 25, 02 5:12 PM Author: Robert Dubner rdubner@dubner.com 201-664-6434
Printed: October 25, 02 5:15 PM ***Bally Gaming, Inc. • 6601 South Bermuda Road • Las Vegas, NV 89119-3605***

2.2. Data Transport Layer

GAT talks to the "slave" EGM over the serial line using the *SVC Serial Protocol*. This protocol is detailed in a GSA document with the following identification information:

Date: May 2, 2000
Document ID: SVC
Version: 01.00.000

A copy of that *SVC Serial Protocol* document should have been found in the same package as this document. Failing that, you can contact GSA at

Gaming Standards Association
39355 California St. Ste #307
Fremont, CA 94538
510.744.4007 (Phone)
510.608.5917 (Fax)
pm@gamingstandards.com (Email)

http://www.gamingstandards.com/

The SVC Protocol was proposed by GSA for exactly this purpose – communicating 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 charater, and doesn't report it to the operator. The philosophy behind that design decision: GAT is a completely interactive program; if the operator doesn't see an expected response, the operator will just 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":

- *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 "Authentication 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 gets back a response from an EGM, the first thing it does is run the entire response through the SHA-1 Secure Hash Algorithm (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.

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 as an ODBC "System DSN" with the name "GATDATA".

The legacy GATDATA database has a very simple heirarchical structure. There can be a number of Manufacturers. Each manufacturer can have associated with it any number of Games. Each game can have associated with it any number of 40-character Hashes.

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 heirarchical 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 Access database, they instead get loaded up with copies of the GATDATA.TXT file, which, obviously, have to be updated from time to time.

When legacy GAT gets back 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 doesn't 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; you 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 like, "THIS VERSION IS OBSOLETE AND THE MACHINE MUST BE UPGRADED."

3. Legacy GAT Works—Why Do We Have To Change It?

We have to change it 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 a Authentication Level 1 Query, the returned response has to be exactly the same for an entire class of machines. Useful information like date stamps, manufacturing dates, serial numbers… all such information *must* be omitted from the Level 1 response, because they change 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. Here is an actual response from a Bally game 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 unacceptible. 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 – The requirements:

- Facilitate the identification of casino games and their components in the field.
- Meet the requirements outlined in Nevada "Proposed Amendments to Technical Standards for Gaming Devices section 1.080 2(d) and
- Meet the requirements outlined in New Jersey TAB01-001.
- 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 Windows 95, Windows 98, Windows 2000, Windows XP. In other words, it has to run on old laptop PCs.
- The GAT program should not require expensive programs or costly upgrades to field agents laptop PCs. 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 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

  Once 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 possiblities 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 acts as conduit between the operator and the game in terms of executing commands, even though it doesn't 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 (that would be me) didn't think of it, then GAT can't 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 by 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 above, GAT3 needs to do many things that the SVC protocol doesn't provide for directly. Thus, GAT3 is reserving 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.

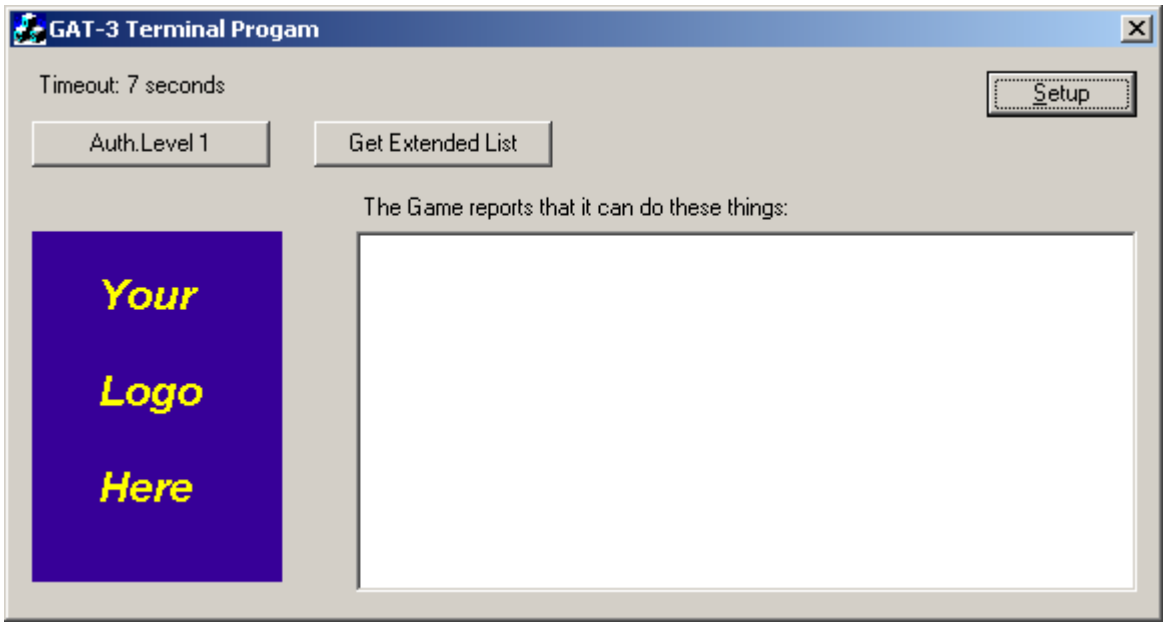
5.1.2. The use of XML in GAT3

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 shortly. If a file is not in XML format, GAT will simply store it and ignore it.

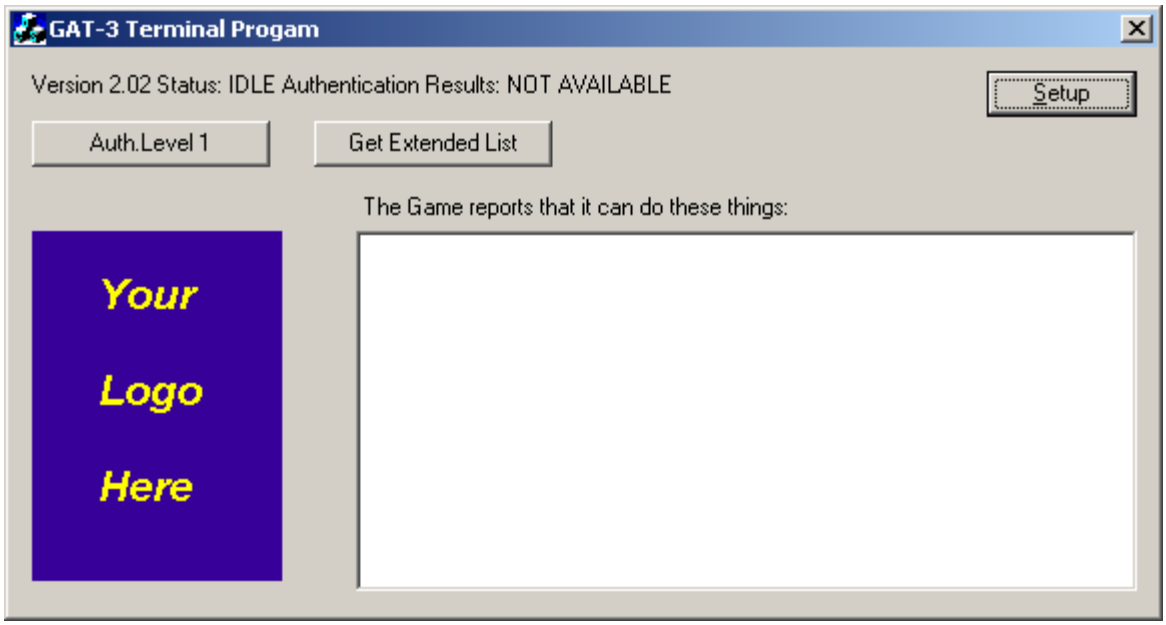
GAT3 is intended for use local to the game. That is, a person will be standing right there, connected to the game when GAT3 is used. However, the designers recognize the advantages of passing GAT3 messages over a network back to a host. By using XML in GAT3 the designers hope to make this dual use of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

5.2. Starting up GAT3 – the "idle" state.

When you start GAT up, it looks like this:



The status line saying, "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 display will look something like this.



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

The various elements of the dialog box are described here:

The Setup button brings up a dialog box that allows for selection of the serial port to use. It also allows the operator to specify where GAT3 is supposed to 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 implemention, the logo of the NJ-DGE is placed there; it is anticipated that various jurisdictions might enjoy doing the same.

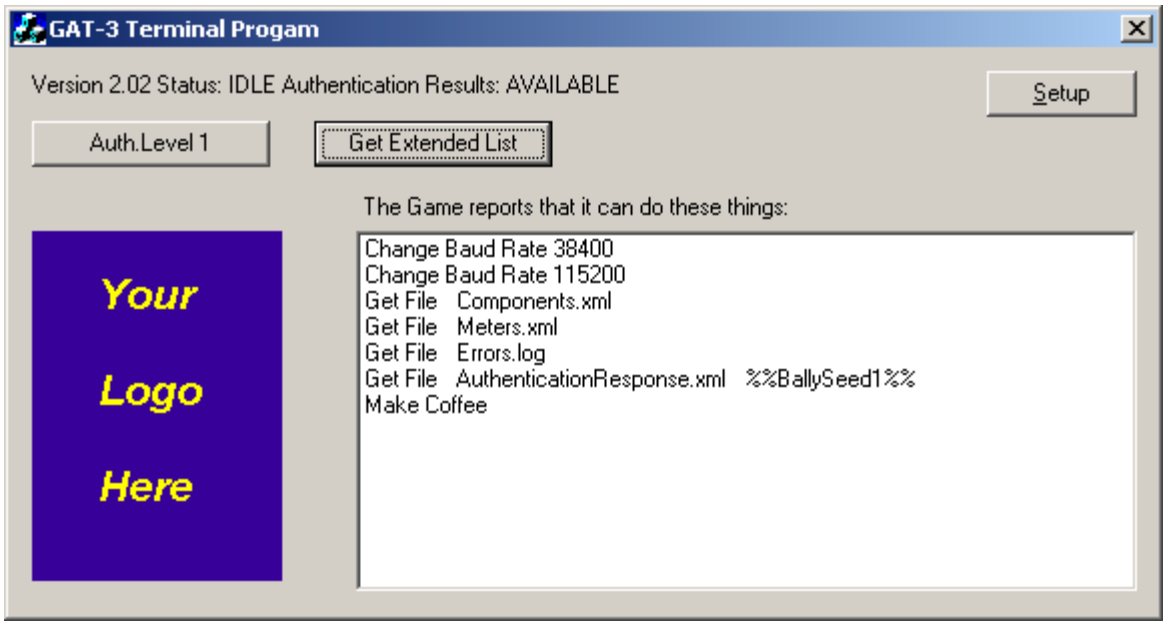
The “Auth. Level 1” button is the legacy GAT button; press it and the program sends out an SVC *Initiate Authentication Calculation Query* with the Authentication Level set to one. The response that comes back is processed as described in the Legacy GAT section up above.

The “Get Extended List” button is how the GAT3 capabilities are accessed.

5.3. The “Get Extended List” button

When the “Get Extended List” button is pressed, GAT3 issues the “Get Special Functions” command to the EGM. The EGM responds with a list of things that it knows how to do. Those things are presented in the GAT3 dialog box so that the user can select them.

Here is a sample of what the results might be when the operator clicks the “Get Extended List” button:



When any of those commands are selected by being double-clicked, GAT3 sends the selected command back to the EGM. As each command is sent, GAT3 examines it to see if it is one of the commands for which special action can be taken. GAT3 knows what to do with the "Change Baud Rate" and "Get File" commands.

Otherwise, GAT3 just 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. Special Commands that GAT3 understands

5.4.1. Change Baud Rate

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

5.4.2. 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.)

5.4.3. Get Special Functions

There is always one additional command that GAT3 understands and will issue to a game: The "Get Special Functions" command. "Get Special Functions" is the command GAT3 issues to the game to get the list shown above.

5.4.4. %% Replacement

Any command can have embedded in it strings of the form seen above, "%%<something>%%. When GAT3 sends a command with one or more of those strings in it, GAT3 looks for a

SEEDS.INI in the same folder as the GAT.EXE executable. It checks a section named “Seeds” 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 that must be shared by all manufacturers, it is suggested that each manufacturer prefix their seed codes with something like “Bally” or “IGT” or “Anchor” or whatever, in order to avoid the collisions that would occur if everybody chose to use the same word, like “Seed”.

5.4.5. 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, but other than that GAT3 will take no special action.

5.5. Getting specific: How the “Get Extended List” button click works.

5.5.1. Sending out the command

When that 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 implicity 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.

5.5.2. What comes back

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 above is seen below:

```
<?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 have to 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

5.6. 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 Function's Feature text 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 up above 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"

5.7. How EGM response files are processed by GAT3

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

5.7.1. The response file is not an XML file

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

5.7.2. The response is an XML file, but there is no GatExec= attribute

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

5.7.3. "Special Functions" response.

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

5.7.4. Default component processing response

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

Here is a default component XML response. It is based on the legacy GAT response shown earlier (the one that demonstrated legacy GAT's flaws). It can be used as a template for GAT3-compliant games:

```
<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 need to 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 Game Name, Manufacturer, Component Name, and Component checksum for each component.

In "laboratory mode", 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 Component Name and Component Checksum, along with the calculated digest, are stored for each component.

In "field mode", 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, the lab can later on 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.

5.7.5. 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 GAT 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. EgmEmulator

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

The EgmEmulator 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 EgmEmulator 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. GAT will process those files exactly as if they had been downloaded from a real game. It is anticipated that the Gat + EgmEmulator combination will prove very useful both to Gat 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.