Encompass® 1-700 Handheld Reader Application Programming Interface

TransCore
8600 Jefferson Street NE
Albuquerque, New Mexico 87113

January 2012

P/N 411885-009

For further information, contact:

TransCore
3410 Midcourt Road, Suite 102
Carrollton, Texas 75006 USA

Phone: (214) 461-4031
Fax: (214) 461-6478

Technical Support

Web: transcore.com/rfidsupport
Phone: (505) 856-8007

For comments or questions about this document, e-mail tech.pubs@transcore.com.

WARNING TO USERS IN THE UNITED STATES

FCC RADIO FREQUENCY INTERFERENCE STATEMENT
47 CFR §15.105(a)

NOTE: This equipment has been tested and found to comply with the limits for a Class A digital device pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency (RF) energy and may cause harmful interference to radio communications if not installed and used in accordance with the instruction manual. Operating this equipment in a residential area is likely to cause harmful interference, in which case, depending on the regulations in effect, the users may be required to correct the interference at their own expense.

NO UNAUTHORIZED MODIFICATIONS
47 CFR §15.21

CAUTION: This equipment may not be modified, altered, or changed in any way without permission from TransCore, LP. Unauthorized modification may void the equipment authorization from the FCC and will void the TransCore warranty.

USE OF SHIELDED CABLES IS REQUIRED
47 CFR §15.27(a)

NOTE: Shielded cables must be used with this equipment to comply with FCC regulations.

TransCore, LP
USA

Contents

Contents

1 Getting Started

2 Overview

3 Library File Functions

A Encompass 1 RFID Scan Handle Specifications

List of Figures

List of Tables

1

Getting Started

Chapter 1

Getting Started

The Encompass®1 Handheld Reader combines the TransCore Encompass 1 RFID Scan Handle and the Intermec Model 700 Color Computer. This application programming interface (API) describes procedures to set up and read tags using TransCore's Encompass 1 Portable RFID Scan Handle with Intermec's Model 700 series Color Computer (model 741, 751, or 761 only). This API provides an overview of the Encompass 1-700 applications interface library kit and describes the library functions.

This guide does not contain detailed user instructions for the mobile computer; those instructions are covered in the Intermec 700 Mobile Computer guides.

Before Getting Started

TransCore's Encompass 1 Handheld Reader is designed to read American Trucking Associations (ATA) tags and Wiegand-programmed tags, and read from and write to TransCore eGo® tags.[1] The TransCore Encompass 1 RFID Scan Handle component (P/N 10-0700-002) is identified by the TransCore logo on the faceplate (Figure 1).

Figure 1-1 Encompass 1 RFID Scan Handle Showing TransCore Logo

Note: *If your Encompass 1 Scan Handle is not identified by the TransCore logo on its faceplate, you will not be able to read ATA tags or TransCore eGo tags.*

1. eGo tags are fully compliant with the American National Standards Institute INCITS 256-2001 and International Organization for Standardization 18000-6 standards.

Handheld Reader Setup Instructions

To set up the basic Encompass 1 Handheld Reader, you need the following items:

- TransCore Encompass 1 Portable 915-MHz Scan Handle with battery
- Intermec 700 Color Mobile Computer (model 741, 751, or 761 only)
- TagReader Demo software program loaded in the Intermec 700 Mobile Computer
- Two #6-32 handle-to-computer retaining screws. These screws are located in the scan handle box in an accessories packet.
- Slotted-head screwdriver and Phillips-head screwdriver
- Scan handle battery pack charger with universal AC adapter and power cord
- 700 Color Mobile Computer single dock/charging station with universal AC adapter and power cord

Installing and Removing the Scan Handle Battery Pack

Before you can use the scan handle, you must charge its battery. The charging process takes approximately four hours.

To remove the battery pack

1. Push the battery release latch forward and remove the battery pack from the handle (Figure 1-2).

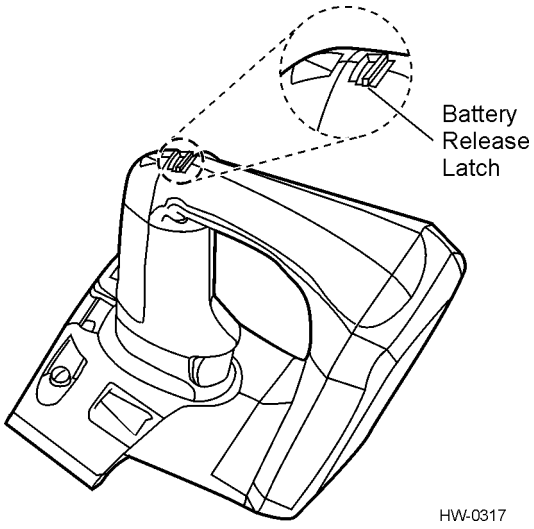

Figure 1-2 Location of Battery Release Latch on Scan Handle

To re-install the battery into the scan handle

The tab at the top of the battery pack fits into the notch in the compartment (Figure 3). Insert the battery (tilt to fit into notch) into the compartment opening and press in until the latch on the bottom of the battery snaps closed.

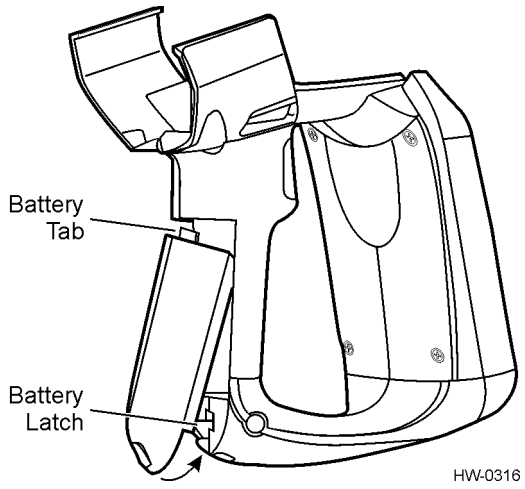

Figure 1-3 Reinserting Battery Pack into Scan Handle

Charging the Scan Handle Battery Pack

The Model Encompass 1 Dual Pack Charger (TransCore P/N 76-0700-002) is suitable for use only with TransCore battery pack (P/N 76-0700-001).

Caution

Use of any other battery pack may present risk of fire or explosion hazard.

Each charger power supply uses approximately 0.5 amps of current.

To charge the battery pack(s)

1. Plug power supply into 110V AC outlet and then connect power supply to charger.
2. Tilt battery pack into position and slip battery pack under lip of charger in compartment. Battery pack tab fits into notch in charger (Figure 1-4).

Note: *Battery pack must be inserted as shown in Figure 1-4.*

3. Push battery pack down and into place.
4. Charge battery pack for four hours.

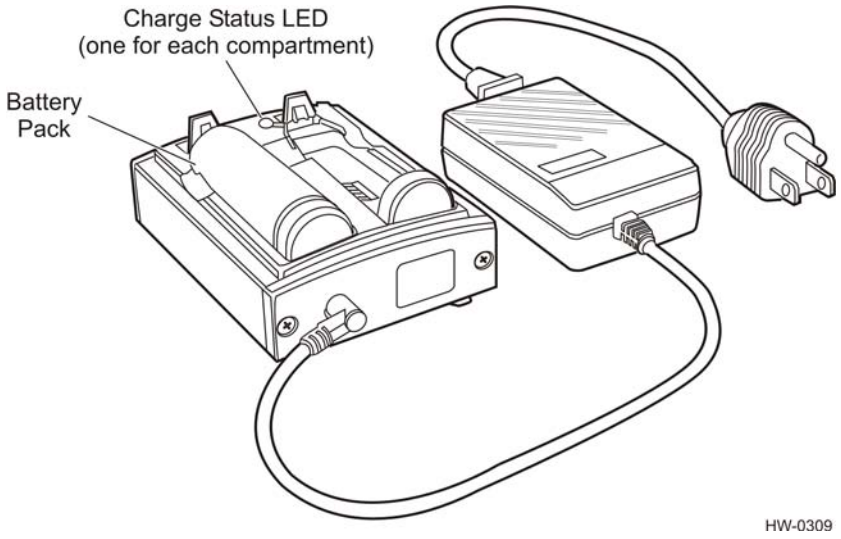

Figure 1-4 Scan Handle Dual Battery Pack Charger

Charging-Status Light-Emitting Diode Indicators

There is one charge status light-emitting diode (LED) for each compartment. The charger LED indicators are as follows:

- Off – Battery not inserted properly, no battery installed, or battery completely run down. The LED remains off until the battery reaches an initial charge capacity of 5%.
- Red – Battery is charging.
- Green – Battery charge is complete.
- Blinking red – pack may not be installed in compartment properly, or pack may be faulty. If pack is installed improperly, remove and reinsert it. If battery has been charging for five hours and the charger LED switches from solid red to blinking red, the pack is faulty and should be replaced.

The LED may flicker as it advances from one charge state to another. When battery is fully charged, remove it from the charger compartment.

Note: *TransCore recommends that you charge the scan handle battery pack after each shift.*

Charging the 700 Color Mobile Computer Battery Pack

Before using the scan handle with the color mobile computer, you need to charge the computer battery using the 700 Mobile Computer Single Dock station.

To charge the mobile computer battery pack using the single dock station

1. Connect power supply unit plug to the single docking station jack (Figure 1-5).

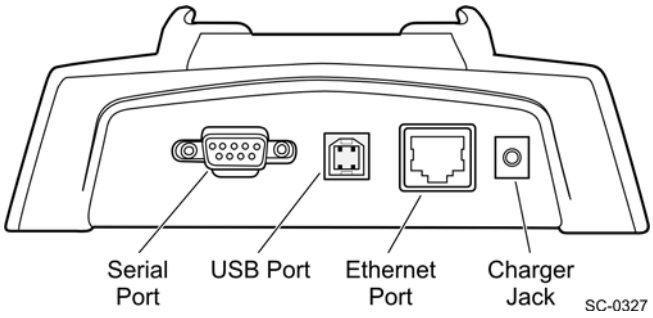

Figure 1-5 Single Dock Station Rear Panel

2. Insert the mobile computer onto the computer docking connector (Figure 1-6).

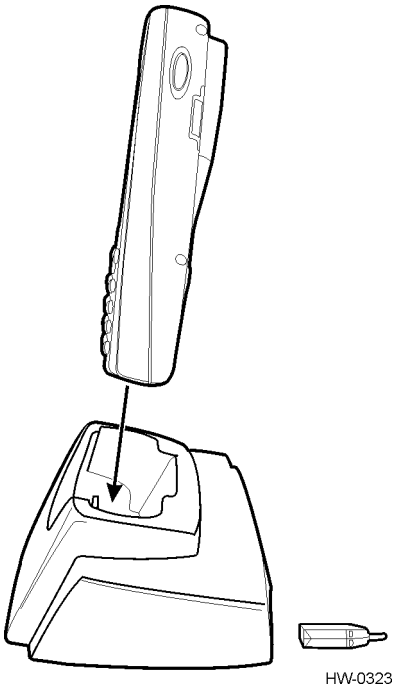

Figure 1-6 Mobile Computer and Single Dock Station

Connecting the Scan Handle to the 700 Color Mobile Computer

Caution

DO NOT remove the gold reflective tape from the inside of the scan handle. This reflective tape aids in infrared data association (IrDA) signal communication between the scan handle and the mobile computer. Your scan handle kit may contain additional reflective tape strips and instructions showing the locations where you should mount the strips.

The mobile computer communicates with the scan handle via IrDA (Figure 1-7). Therefore, the IrDA signal passing from the scan handle must meet with the IrDA window on the mobile computer and must create an acceptable connection.

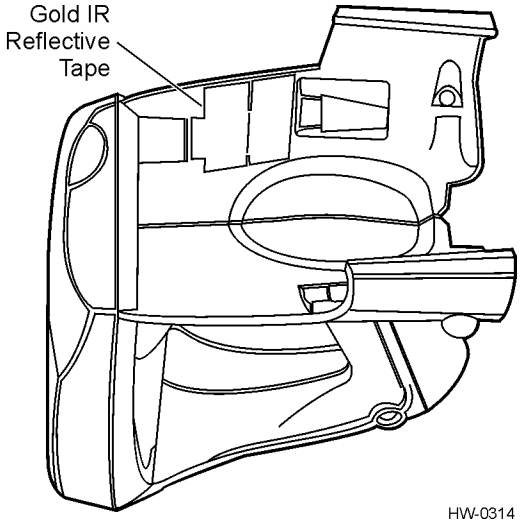

Figure 1-7 Location of Gold IrDA Reflective Tape in Scan Handle

To connect the scan handle and mobile computer

1. Remove two Phillips-head screws from the mobile computer shown in Figure 8. Keep the Phillips-head screws handy in case you need to re-install them later. Slide the Intermec 700 Series Mobile Computer into your scan handle. Ensure that the computer slides all the way into the scan handle. You hear a click when you have the mobile computer fully inserted.

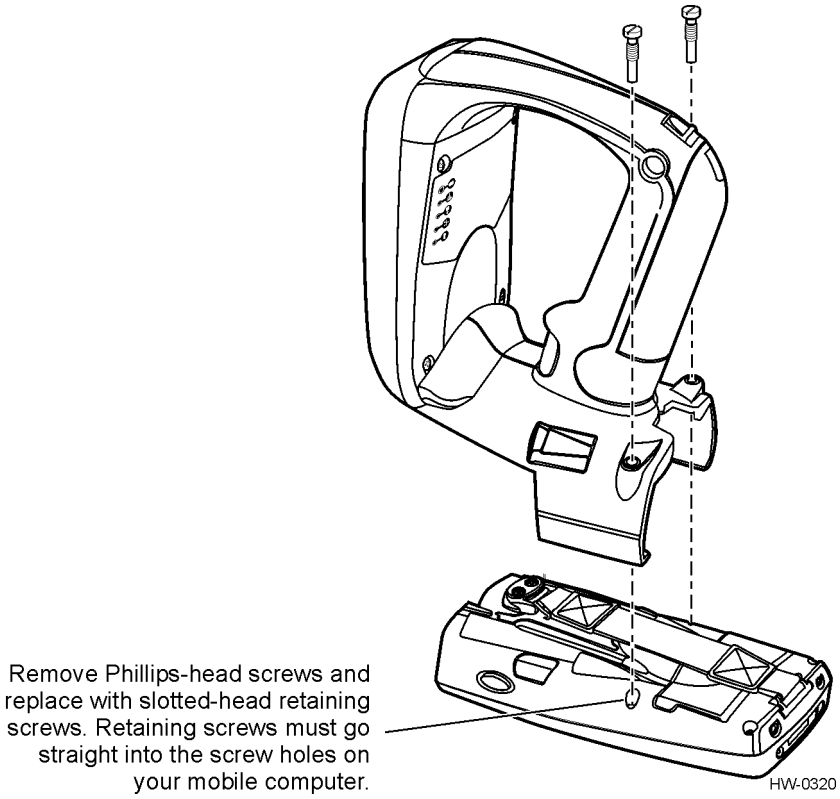

Figure 1-8 Connecting Intermec Series 700 Mobile Computer to Scan Handle

2. Remove the two slotted-head retaining screws from the accessories bag. These retaining screws are easy to identify because they are only partly threaded.
3. Insert and tighten the two slotted-head retaining screws through the scan handle and into the screw holes in the mobile computer (Figure 1-8).

Note: *The computer is not secured to the scan handle unless the retaining screws are correctly aligned and tightened.*

Figure 1-9 shows the fully assembled scan handle and mobile computer.

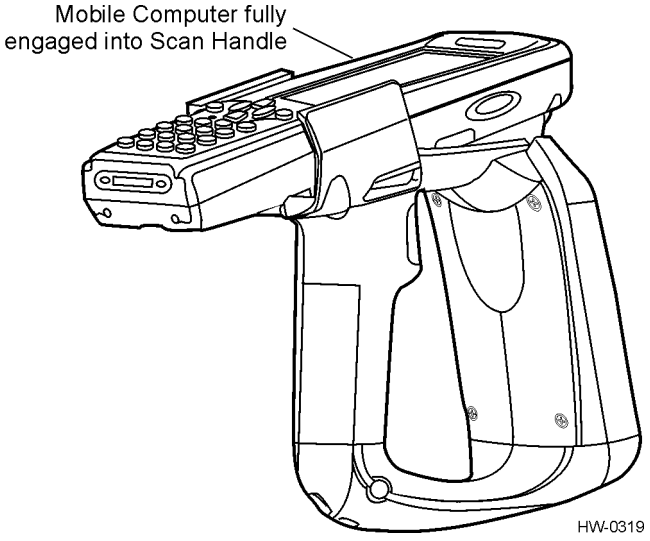

Figure 1-9 Intermec 700 Mobile Computer Fully Inserted into Scan Handle

To remove the mobile computer from the scan handle

1. Remove the two retaining screws.
2. Grasp the 700 Series Color Mobile Computer at the display end, and pull forward toward the handle face. Replace the two Phillips-head screws if needed.

Testing the Encompass 1 Handheld Reader

Refer to the *Intermec 700 Mobile Computer User Guide* to learn about the 700 Color Mobile Computer features and functions.

Note: *Ensure that the mobile computer battery is charged before proceeding.*

1. Switch on the mobile computer by pressing the yellow I/O button. The first time you switch on your mobile computer, it boots to the operating system. After a few seconds, you see the Windows Mobile 2003 Welcome screen. Tap the screen with your stylus to advance to the next display on the screen.
2. You are prompted through several screens to complete the setup process. Read the display messages and follows the instructions. When you reach the Windows Mobile Today screen, you have completed the computer operating system setup. After the computer operating system is reset, the TagReader Demo software re-installs.
3. Once the TagReader Demo program is re-installed, use your stylus to select the **TagReader Demo** icon or name listed in the Start Menu (Figure 1-10).

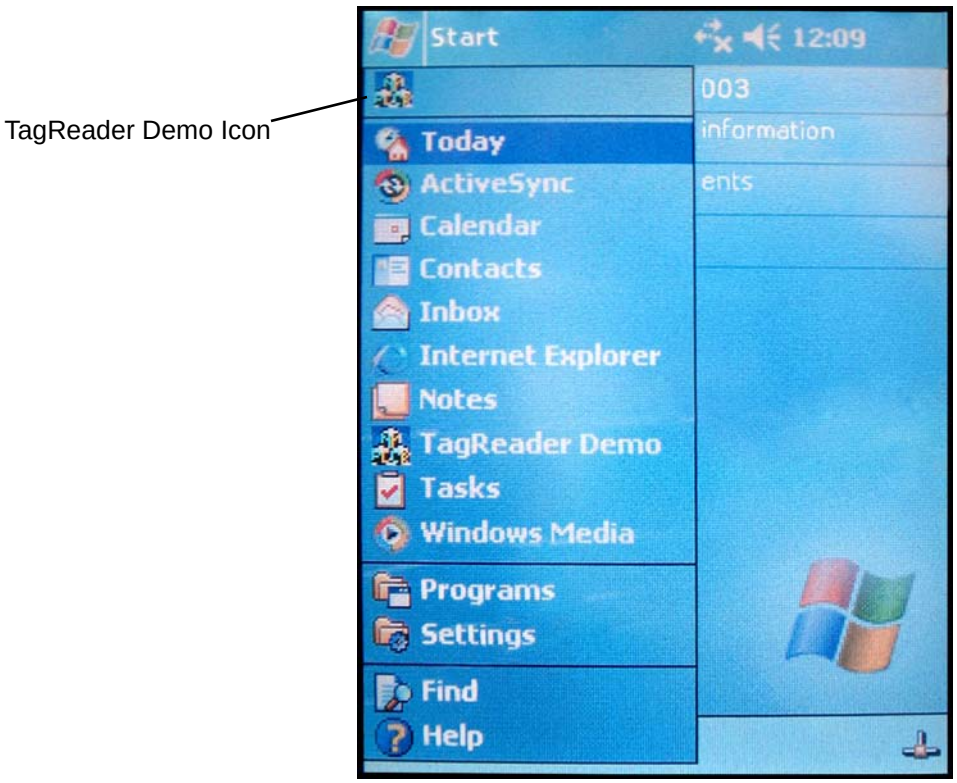

Figure 1-10 TagReader Demo Icon and Program Listed in Start Menu

4. Squeeze scan handle trigger (Figure 1-11) to check battery capacity as indicated on bottom LED (Figure 1-12 and Table 1-1).

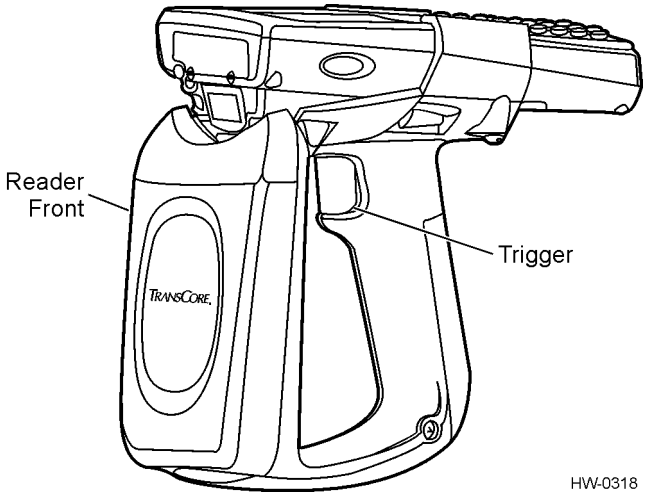

Figure 1-11 Location of Trigger

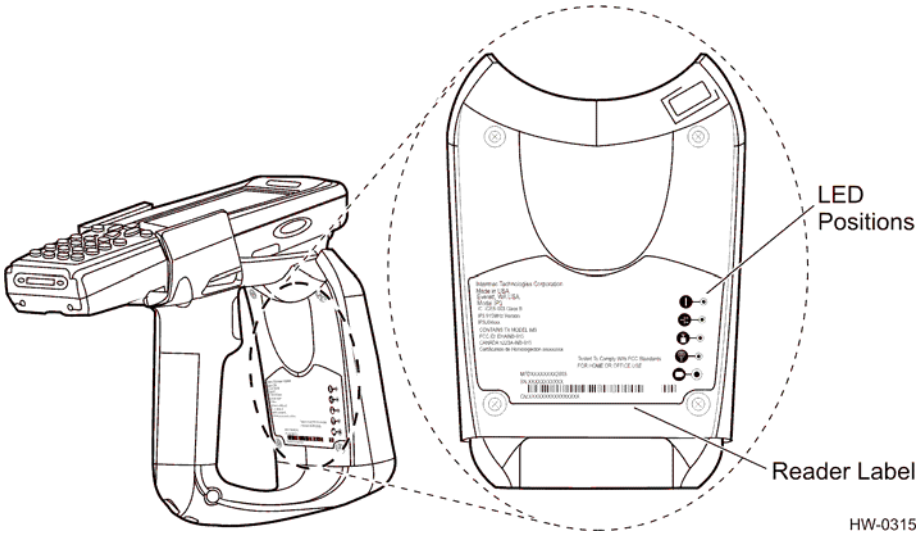

Figure 1-12 Location of Scan Handle LEDs

Scan Handle LED Locations and Definitions

Table 1-1 lists the scan handle LEDs and corresponding symbols.

Table 1-1 Scan Handle LED Descriptions

LED Symbol	Indication	Description
	PWR	+5V DC is on and the scan handle is ready for use.
	HOST COMM	Data communication with host is active.
	RF ON	RF power is on.

Table 1-1 Scan Handle LED Descriptions (continued)

LED Symbol	Indication	Description
	TAG COMM	Valid tag transaction
	BATT	Battery condition: Red – 0% to 20% charge Orange – 20% to 80% charge Green – 80% to 100% charge

Reading Tags

Note: *If your Encompass 1 Scan Handle is not identified by the TransCore logo on its faceplate, you will not be able to read ATA tags and TransCore eGo tags.*

To read tags

1. Start the mobile computer **TagReader Demo** program by pressing the input/output (I/O) button on the computer.
2. Select the **TagReader Demo** icon or Program Name in the Start Menu (Figure 1-10).
3. Aim the scan handle at the RFID tag and squeeze the trigger. To obtain a good first-time read, you need to be within three feet (one meter) of the tag.
4. Check the 700 Mobile Computer screen to see if the tag type and ID displayed.

   The scan handle reads ATA (full-frame), Wiegand (half-frame), or eGo tags, you do not need to know which type of tag you are reading.

An example of a screen display for an ATA tag transaction is shown in Figure 1-13.

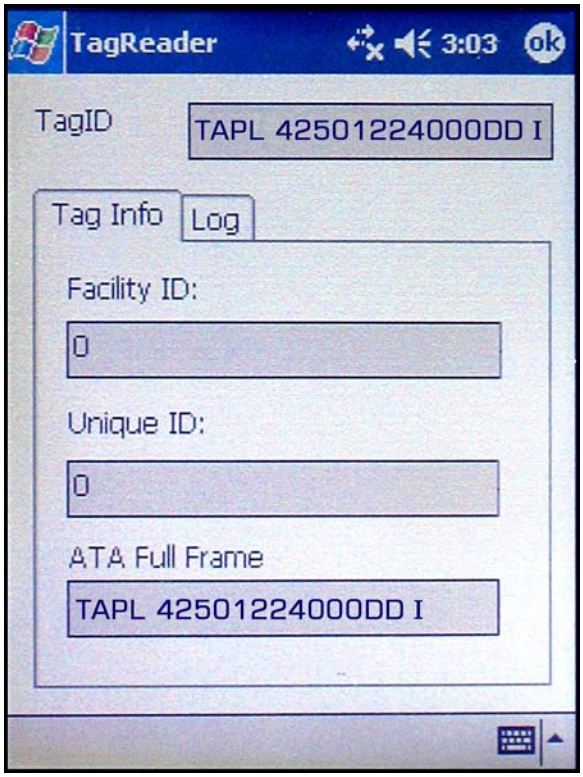

Figure 1-13 Sample Display of ATA Tag Read

An example of a screen display for a half-frame Wiegand tag read is shown in Figure 1-14.

Figure 1-14 Sample Display of Wiegand Tag Read

An example of a screen display for an eGo tag read is shown in Figure 1-15.

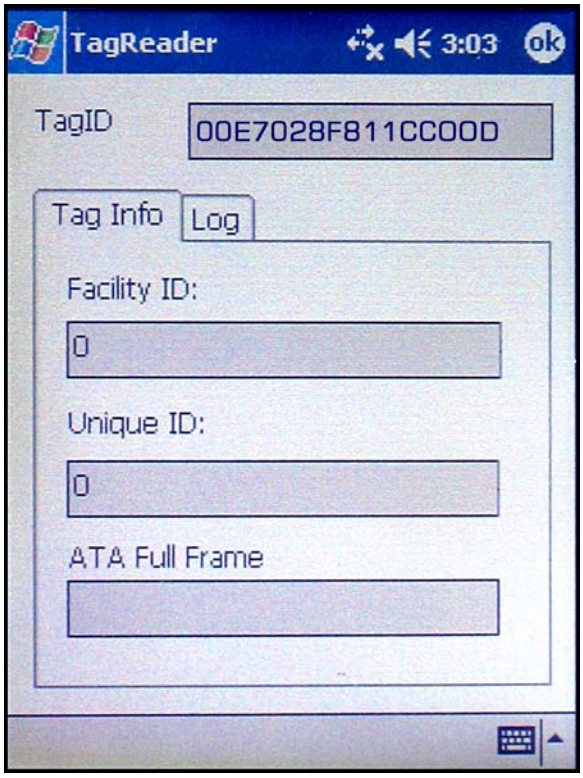

Figure 1-15 Sample Display of eGo Tag Read

Resetting Your 700 Color Mobile Computer

Applications operating on your 700 Color Mobile Computer may stop responding to system commands. Should this occur, you may need to perform either a warm reset or cold reset of your computer. Refer to the warm and/or cold resetting sections in the Intermec *700 Color Mobile Computer With Windows Mobile 2003 Quick Start Guide*.

2

Overview

Chapter 2

Overview

This chapter provides an overview of the Encompass® 1-700 applications interface library kit.

TransCore Encompass 1-700 Application Programming Interface Library Kit

The application programming interface kit provides three libraries: ***IP3Reader.lib***, ***ParseATA.lib***, and ***ParseAAR.lib***.

The ***IP3Reader.lib*** library provides simple functions to easily open or close connections to an Encompass 1 reader and also read raw tag data using the Encompass 1 reader. Additionally, the read function contained in this library provides return result code information to determine whether the read was successful. The read function identifies which type of data was read: eGo® tag ID data, Wiegand-formatted data, or ATA-formatted data[1].

The ***ParseATA.lib*** library provides simple functions to decode ATA data from a raw packet of ATA-formatted data that has been read using the Encompass 1 reader.

The ***ParseAAR.lib*** library provides simple functions to decode AAR data from a raw packet of AAR-formatted data that has been read using the Encompass 1 reader.

The ***IP3Utils.Dll*** DLL is a .Net class that provides .Net wrappers for functions in ***IP3Reader.lib***, ***ParseATA.lib***, and ***ParseAAR.lib***.

For a typical application, the developer would first call the IP3Open() function from the ***IP3Reader.lib*** library in order to open a connection to the reader.

Once a connection to the Encompass 1 reader is open, the developer waits for the user to pull the trigger on the Encompass 1 reader sled. (Examples of how to detect that the Encompass 1 reader's trigger has been pulled are shown in the sample application code titled TagReader Demo code.)

When the application detects that the Encompass 1 reader's trigger has been pulled, the developer issues the IP3Identify() command to the reader. The developer then examines the result code returned from the IP3Identify() command to determine the results of the read request.

If the IP3Identify() command's result code indicates that a failure occurred, the devel-

1. If ATA-formatted data is returned from a call to the IP3Identify() function, the application developer should then use the IsAAR() function provided in the ParseAAR.lib library to determine if the ATA data packet contains AAR data.

oper should close and then re-open the reader connection before making any further calls to the IP3Identify() command.

If the IP3Identify() command's result code indicates that no data was read, the developer tells the user that the read attempt was unable to read any tag data and returns to waiting for the user to pull the trigger again.

If the IP3Identify() command's result code does not indicate a failure or no data, then the developer examines the specific value of the result code to determine the type of data that was successfully read. This examination is important because the IP3Identify() command always return only the raw data that was actually read from a tag. This data must then be decoded using the appropriate decode commands for the type of data that is contained in the raw data packet that was returned from the IP3Identify() command.

IP3Reader Library Functions

```
int AtaIP3ReaderOpen(void);

int AtaIP3ReaderClose(void);

int AtaIP3Identify(char *pTagId);

int AtaIP3ReadEgoAta(const char *pTagId, char *pAtaData);
```

ParseATA Library Functions

```
int DecodeATAData(const char *pTagData, char *pATACharacters,
int FrameSize);
int DecodeFirstCheckSum(const char *pTagData);
int DecodeHalfFrameMarker(const char *pTagData);
int DecodeSecondCheckSum(const char *pTagData);
int DecodeFullFrameMarker(const char *pTagData);
```

ParseAAR Library Functions

```
/* AAR General */

int IsAAR(const char *pTagData);

int DecodeEquipmentGroup(const char *pTagData);

int DecodeTagType(const char *pTagData);

void DecodeEquipmentInitial(const char *pTagData,
                            char *pEquipmentInitials);

int DecodeReserved(const char *pTagData);

int DecodeSecurityBits(const char *pTagData);

void DecodeSecurity(const char *pTagData,
                    char *pSecurityChars); /* 3 char buffer */
```

```
int DecodeDataFormat(const char *pTagData);

/* AAR RAILCAR specific (EGC_RAILCAR) */
int DecodeRailcarNumber(const char *pTagData);
int DecodeRailcarSideIndicator(const char *pTagData);
int DecodeRailcarNumberOfAxles(const char *pTagData);
int DecodeRailcarBearingType(const char *pTagData);
int DecodeRailcarLength(const char *pTagData);  /* decimeters
*/
int DecodeRailcarPlatformIdentifier(const char *pTagData);
int DecodeRailcarSpare(const char *pTagData);

/* AAR Locomotive specific (EGC_LOCOMOTIVE) */
int DecodeLocomotiveNumber(const char *pTagData);
int DecodeLocomotiveSideIndicator(const char *pTagData);
int DecodeLocomotiveNumberOfAxles(const char *pTagData);
int DecodeLocomotiveBearingType(const char *pTagData);
int DecodeLocomotiveLength(const char *pTagData);  /* decime-
ters */
int DecodeLocomotiveSpare(const char *pTagData);

/* AAR Trailer specific (EGC_TRAILER) */
void DecodeTrailerNumber(const char *pTagData,
                         char *pTrailerNumber); /* 9 char buffer
*/
int DecodeTrailerLength(const char *pTagData); /* centimeters
*/
int DecodeTrailerWidth(const char *pTagData);  /*
TANDEM_WIDTH_CODES */
int DecodeTrailerTandemWidth(const char *pTagData); /
*TANDEM_WIDTH_CODES*/
int DecodeTrailerTypeDetail(const char *pTagData); /*
TRAILER_TYPE_DETAIL_CODES */
int DecodeTrailerForwardExtension(const char *pTagData);
int DecodeTrailerTareWeight(const char *pTagData);   /* 100 kg
*/
```

```
int DecodeTrailerHeight(const char *pTagData);

/* AAR Chassis specific (EGC_CHASSIS) */
int DecodeChassisNumber(const char *pTagData);
int DecodeChassisTypeDetail(const char *pTagData);
int DecodeChassisTareWeight(const char *pTagData);
int DecodeChassisHeight(const char *pTagData);
int DecodeChassisTandemWidth(const char *pTagData);
int DecodeChassisForwardExtension(const char *pTagData);
int DecodeChassisKingpinSetting(const char *pTagData);
int DecodeChassisAxleSpacing(const char *pTagData);
int DecodeChassisRunningGearLocation(const char *pTagData);
int DecodeChassisNumberOfLengths(const char *pTagData);
int DecodeChassisMinimumLength(const char *pTagData);
int DecodeChassisSpare(const char *pTagData);
int DecodeChassisMaximumLength(const char *pTagData);

/* AAR End-of-train device specific */
int DecodeEndOfTrainNumber(const char *pTagData);
int DecodeEndOfTrainType(const char *pTagData);
int DecodeEndOfTrainSideIndicator(const char *pTagData);
int DecodeEndOfTrainSpare1(const char *pTagData);
int DecodeEndOfTrainSpare2(const char *pTagData);

/* AAR Intermodal Container specific */
int DecodeIntermodalNumber(const char *pTagData);
int DecodeIntermodalCheckDigit(const char *pTagData);
int DecodeIntermodalLength(const char *pTagData);
int DecodeIntermodalHeight(const char *pTagData);
int DecodeIntermodalWidth(const char *pTagData);
int DecodeIntermodalContainerType(const char *pTagData);
int DecodeIntermodalMaximumGrossWeight(const char *pTagData);
int DecodeIntermodalTareWeight(const char *pTagData);
```

```
int DecodeIntermodalSpare(const char *pTagData);

/* AAR Multimodal Equipment specific */
int DecodeMultimodalNumber(const char *pTagData);
int DecodeMultimodalSideIndicator(const char *pTagData);
int DecodeMultimodalNumberOfRailAxles(const char *pTagData);
int DecodeMultimodalBearingType(const char *pTagData);  /*
BEARING_TYPE_CODES */
int DecodeMultimodalLength(const char *pTagData)
int DecodeMultimodalPlatformIdentifier(const char *pTagData);
int DecodeMultimodalTypeDetail(const char *pTagData);   /*
TRAILER_TYPE_DETAIL_CODES */
int DecodeMultimodalSpare(const char *pTagData);
```

IP3UTILL.DLL Methods

```
public IP3Scanner();
public void InitIP3Scanner();

public int Identify(out string TagId);
public int ReadEgoAta(ref string TagId, out string AtaData);

public int IP3DecodeFirstCheckSum(string pTagData);
public int IP3DecodeHalfFrameMarker(string pTagData);
public int IP3DecodeSecondChecksum(string pTagData);

public int IP3DecodeFullFramerMarker(string pTagData);
public int IP3Is26BitWiegand(ref string pTagData);
public int IP3Is34BitWiegand(string pTagData);
public int IP3Is37BitMcGain(string pTagData);
public int IP3DecodeAtaData(string pTagData,
```

```
                out string pAtaData, int FrameSize);
public int IP3Decode26BitWiegand(string pTagData,
                ref int pFacility, ref int pUniqueID);
public int IP3Decode37BitMcGain(string pTagData,
                ref int pFacility, ref int pUniqueID);

public int IP3IsAAR(string pTagData);
public int IP3DecodeEquipmentGroup(string pTagData);
public int IP3DecodeTagType(string pTagData);
public void IP3DecodeEquipmentInitial(string pTagData,out
string pInititals);
public int IP3DecodeReserved(string pTagData);
public int IP3DecodeSecurityBits(string pTagData);
public void DecodeSecurity(string pTagData,
                out string pSecurityChars);
public int IP3DecodeDataFormat(string pTagData);

public int IP3DecodeRailcarNumber(string pTagData);
public int IP3DecodeRailcarSideIndicator(string pTagData);
public int IP3DecodeRailcarNumberOfAxles(string pTagData);
public int IP3DecodeRailcarBearingType(string pTagData);
public int IP3DecodeRailcarLength(string pTagData);
public int IP3DecodeRailcarPlatformIdentifier(string pTagData);
public int IP3DecodeRailcarSpare(string pTagData);
public int IP3DecodeLocomotiveNumber(string pTagData);
public int IP3DecodeLocomotiveSideIndicator(string pTagData);
public int IP3DecodeLocomotiveNumberOfAxles(string pTagData);
public int IP3DecodeLocomotiveBearingType(string pTagData);
```

```
public int IP3DecodeLocomotiveLength(string pTagData);
public int IP3DecodeLocomotiveSpare(string pTagData);
public int IP3DecodeTrailerNumber(string pTagData);
public int IP3DecodeTrailerLength(string pTagData);
public int IP3DecodeTrailerWidth(string pTagData);
public int IP3DecodeTrailerTandemWidth(string pTagData);
public int IP3DecodeTrailerTypeDetail(string pTagData);
public int IP3DecodeTrailerForwardExtension(string pTagData);
public int IP3DecodeTrailerTareWeight(string pTagData);
public int IP3DecodeTrailerHeight(string pTagData);
public int IP3DecodeChassisTypeDetail(string pTagData);
public int IP3DecodeChassisTareWeight(string pTagData);
public int IP3DecodeChassisHeight(string pTagData);
public int IP3DecodeChassisTandemWidth(string pTagData);
public int IP3DecodeChassisForwardExtension(string pTagData);
public int IP3DecodeChassisKingpinSetting(string pTagData);
public int IP3DecodeChassisAxleSpacing(string pTagData);
public int IP3DecodeChassisRunningGearLocation(string
pTagData);
public int IP3DecodeChassisNumberOfLengths(string pTagData);
public int IP3DecodeChassisMinimumLength(string pTagData)
public int IP3DecodeChassisSpare(string pTagData);
public int IP3DecodeChassisMaximumLength(string pTagData);
public int IP3DecodeEndOfTrainNumber(string pTagData);
public int IP3DecodeEndOfTrainType(string pTagData);
public int IP3DecodeEndOfTrainSideIndicator(string pTagData);
public int IP3DecodeEndOfTrainSpare1(string pTagData);
public int IP3DecodeEndOfTrainSpare2(string pTagData);
```

```
public int IP3DecodeIntermodalNumber(string pTagData);

public int IP3DecodeIntermodalCheckDigit(string pTagData);

public int IP3DecodeIntermodalLength(string pTagData);

public int IP3DecodeIntermodalHeight(string pTagData);

public int IP3DecodeIntermodalWidth(string pTagData);

public int IP3DecodeIntermodalContainerType(string pTagData);

public int IP3DecodeIntermodalMaximumGrossWeight(string
pTagData);

public int IP3DecodeIntermodalTareWeight(string pTagData);

public int IP3DecodeIntermodalSpare(string pTagData);

public int IP3DecodeMultimodalNumber(string pTagData);

public int IP3DecodeMultimodalSideIndicator(string pTagData);

public int IP3DecodeMultimodalNumberOfRailAxles(string
pTagData);

public int IP3DecodeMultimodalBearingType(string pTagData);

public int IP3DecodeMultimodalLength(string pTagData);

public int IP3DecodeMultimodalPlatformIdentifier(string
pTagData);

public int IP3DecodeMultimodalTypeDetail(string pTagData);

public int IP3DecodeMultimodalSpare(string pTagData);
```

3

Library File Functions

Chapter 3

Library File Functions

This chapter describes the Encompass® 1 Scan Handle library functions.

IP3Reader Library Functions

This section describes the IP3Reader Library Functions. Each function's section lists the code, the value returned, any parameters, any remarks, and an example of the code.

Note: *This API provides functions that manipulate buffer arrays containing ANSI characters (ASCII). ANSI character strings cannot be directly displayed by applications operating on Windows CE-based platforms. Windows CE applications can only display Unicode character strings. To maintain portability with desktop applications, the simplest way to convert an ANSI string for display is to use MFC's CString Format method using a format specifier _T("%hc") for a single ANSI character, or _T("%hs") for an ANSI nul-terminated string. A non-MFC equivalent can be done by declaring a TCHAR arrary and using the _sntprintf function with the same format specification syntax.The _sntprintf function is preferred over _stprintf because it allows size specification for the destination buffer, which avoids memory overflow errors.*

IP3ReaderOpen

Open connection to the Encompass 1 reader device.

int **IP3ReaderOpen(void);**

Return Value

This function returns a non-zero value if a connection if there is a problem opening the connection to the reader. If the reader connection is successfully opened, this function returns a value of zero.

Parameters

none

Remarks

none

Example

```
//Open the IP3 reader
      if (!IP3ReaderOpen())
    {
      MessageBox(_T("Failed to open Reader. Make sure the bat-
```

```
tery is Okay and IP3 is properly seated"),_T("IP3 Error"));

    }
```

IP3ReaderClose

Close an open Encompass 1 reader connection.

int **IP3ReaderClose(void);**

Return Value

This function returns a non-zero value if there is a problem closing the connection to the reader. If the reader connection is successfully closed, this function returns a value of zero.

Parameters

none

Remarks

none

Example

```
BOOL CTagReaderDlg::DestroyWindow()

{
      I700S_HwEvents_Shutdown(); // Close 700HwEvents interface

      IP3ReaderClose();

      return CDialog::DestroyWindow();
}
```

IP3Identify

Use an open Encompass 1 reader connection to command the reader device to attempt to identify a tag if present.

int **IP3Identify(char ****pTagData* **);**

Return Value

A return value of eTT_None indicates that no tag data was read.

A return value of eTT_Ego indicates that an eGo® tag was identified and the tag contains no specially formatted data. In this case, *pTagData* contains the eGo tag's eight byte serial number.

A return value of eTT_SeGo indicates that an eGo Plus Sticker Tag (PST) was identified and the tag contains no specially formatted data. In this case, pTagData contains the eGo PST's identification number.

A return value of eTT_ATA_HALF indicates that *pTagData* contains Wiegand-formatted data.

A return value of eTT_ATA indicates that *pTagData* contains ATA-formatted data.

A return value of eTT_Error indicates that a communications error occurred.

Parameters

pTagData

Pointer to a buffer where the function places the tag data on success. This buffer should should be allocated for at least 16 * 2 hexadecimal characters per byte + 1 for the nul string terminator. The constant *MAX_IDENTIFY_CHARS* defined in the header should be used when declaring the buffer.

Remarks

A return value of eTT_Error indicates that a communications error occurred. If this happens, close and then re-open the connection to the reader before trying to perform any further identify or read operations.

Example

```
int RC = IP3Identify(m_TagId);
char m_Tagid[MAX_IDENTIFY_CHARS];
```

IP3ReadEgoAta

Use an open Encompass 1 reader connection to read the ATA data frame from an eGo tag.

int **IP3ReadEgoAta(const char *** *pTagData*, **const char *** *pAtaData* **);**

Return Value

Positive return value indicates success.

A zero return value indicates a read error.

A negative return value indicates a communications error occurred.

Parameters

pTagData

Pointer to a buffer containing the tag data that was returned from a previous call to the *IP3Identify()* function in which the function return code indicated that the data read was eGo Tag ID data.

pAtaData

Pointer to a buffer where the function places the tag data on success. This buffer should be allocated at least 33 bytes (16 tag data bytes * 2 hexadecimal characters each + 1 for the nul string terminator). The constant *MAX_IDENTIFY_CHARS* defined in the header file should be used when declaring the buffer.

Remarks

A negative return value indicates that a communications error occurred. If this happens, close and then re-open the connection to the reader before trying to perform any further identify or read operations.

Example

```
int RC = IP3ReadEgoAta(m_TagId, m_TagAtaData);
char m_TagAtaData[MAX_IDENTIFY_CHARS];
```

ParseATA Library Functions

This section describes the ParseATA Library Functions. Each function's section lists the code, the value returned, any parameters, any remarks, and an example of the code.

DecodeATAData

Decode the 20 ATA six-bit ASCII characters contained in the ATA-formatted data specified by the *pTagData* parameter.

int **DecodeATAData (const char ****pTagData*, **char ****pATACharacters*, *int FrameSize* **);**

The FrameSize parameter controls the number of characters that are converted. The expected values for this parameter are constants defined in the IP3Reader.h header file:

ATA_HALF_FRAME_SIZE — Used when decoding only 10 six-bit characters from half-frame ATA tags

ATA_FULL_FRAME_SIZE — Used when decoding all 20 six-bit characters from full-frame ATA tags

Return Value

This function returns 20 eight-bit ASCII characters, along with a terminating nul character, in the buffer space pointed to by the *pATAData* parameter.

The return value indicates the success or failure of the conversion, result code values are defined in the ParseATA.h header:

eATA_SUCCESS — Success

eATA_NULLPARAMERROR — Required parameter was a NULL pointer. Conversion cannot proceed.

eATA_CHECKSUMERROR — The buffer pointed to by *pTagData* does not pass checksum validation, character conversion not performed.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

pATACharacters

Pointer to a buffer that contains the ATA characters when the function returns.

Remarks

This function only should be used on ATA tag data that is not AAR-formatted data.

Example

```
DecodeAtaData(m_TagId, Buffer, FrameSize);
```

DecodeFirstCheckSum

Decode the value of the first checksum field contained in the ATA-formatted data specified by the *pTagData* parameter.

int **DecodeFirstCheckSum(const char ***pTagData* **);**

Return Value

This function returns the value of the first checksum field. Valid values are 0, 1, 2, or 3.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
memcpy(m_TagAtaData, m_TagId, sizeof m_TagAtaData);
DecodeFirstCheckSum(m_TagAtaData);
```

DecodeHalfFrameMarker

Decode the value of the first frame marker field (also called the half frame marker field) contained in the ATA-formatted data specified by the *pTagData* parameter.

int **DecodeHalfFrameMarker(const char ****pTagData* **);**

Return Value

This function returns the value of the first frame marker field. A value of 2 indicates that only the first half ATA frame contains usable data. A value of 4 indicates that both the first and second half ATA frames contain usable data.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
memcpy(m_TagAtaData, m_TagId, sizeof m_TagAtaData);
DecodeHalfFrameMarker(m_TagAtaData);
```

DecodeSecondCheckSum

Decode the value of the second checksum field contained in the ATA-formatted data specified by the *pTagData* parameter.

int **DecodeSecondCheckSum(const char ***pTagData* **);**

Return Value

This function returns the value of the second checksum field. Valid values are 0, 1, 2, or 3.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
memcpy(m_TagAtaData, m_TagId, sizeof m_TagAtaData);
DecodeSecondCheckSum(m_TagAtaData);
```

DecodeFullFrameMarker

Decode the value of the second frame marker field (also called the full frame marker field) contained in the ATA-formatted data specified by the *pTagData* parameter.

int **DecodeFullFrameMarker(const char ***pTagData* **);**

Return Value

This function returns the value of the second frame marker field.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
memcpy(m_TagAtaData, m_TagId, sizeof m_TagAtaData);
DecodeFullFramMarker(m_TagAtaData);
```

ParseAAR Library Functions

This section describes the ParseAAR Library Functions. Each function's section lists the code, the value returned, any parameters, any remarks, and an example of the code.

IsAAR

Determine if data contained in a buffer is AAR-formatted data.

int IsAAR(const char **pTagData*);

Return Value

This function returns a value of 1 if the data contained in *pTagData* is AAR-formatted data, otherwise, the function returns a value of 0.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
int iAAR = IsAAR(m_TagAtaData);
```

DecodeEquipmentGroup

Decode the equipment group code value contained in the AAR-formatted data block provided in the *pTagData* parameter.

int DecodeEquipmentGroup(const char ***pTagData* **);

Return Value

This function returns the value contained in the equipment group code field. Valid range of values is 0-31. For freight and passenger railcars, this value is decimal 19.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
int groupCode = DecodeEquipmentGroup(m_TagAtaData);
```

DecodeTagType

Decode the tag type value contained in the AAR-formatted data block provided in the *pTagData* parameter.

int DecodeTagType(const char **pTagData*);

Return Value

This function returns the value contained in the tag type field. This value always equals decimal two (2). If this value is equal to anything other than decimal two, do not use these functions for any further decoding of the data contained in *pTagData*.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
int tagType = DecodeTagType(m_TagAtaData);
```

DecodeEquipmentInitial

Decode the equipment initial value contained in the AAR-formatted data block provided in the *pTagData* parameter.

int DecodeEquipmentInitial(const char ****pTagData*, **char ****pEquipmentInitials* **);

Return Value

This function decodes the four-character equipment initial data contained in the AAR-formatted data provided by the *pTagData* parameter and returns the equipment initial characters in the *pEquipmentInitials* parameter.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

pEquipmentInitials

Pointer to a buffer that contains the Equipment Initial characters on successful function completion.

Remarks

none

Example

```
DecodeEquipmentInitial(m_TagAtaData, Buffer);
```

DecodeReserved

Decode the Reserved value contained in the AAR-formatted data block provided in the *pTagData* parameter.

int **DecodeReserved(const char **pTagData*);**

Return Value

This function returns the reserved field of an AAR-formatted tag.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
DecodeReserved(m_TagAtaData);
```

DecodeSecurityBits

Decode the security bits information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeSecurityBits(const char ****pTagData*, **char ****pSecurityChars* **);**

Return Value

This function returns the security bits in an AAR-formatted tag.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

pSecurityChars

Pointer to a buffer that contains the two security characters (plus a nul character) when the function returns.

Remarks

none

Example

```
DecodeSecurityBits(m_TagAtaData, Buffer);
```

DecodeSecurity

Decode the security bits information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeSecurity(const char *** *pTagData*, **char *** *pSecurityChars* **);**

Return Value

This function decodes the four-character security data contained in the AAR-formatted data provided by the *pTagData* parameter and returns the security characters in the *pSecurityChars* parameter.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

pSecurityChars

Pointer to a buffer that contains the two security characters (plus a nul character) when the function returns.

Remarks

none

Example

```
DecodeSecurity(m_TagAtaData, Buffer);
```

DecodeDataFormat

Decode the data format information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeDataFormat(const char ****pTagData* **);**

Return Value

This function returns the data format of an AAR-formatted tag.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
int DataFormatCode = DecodeDataFormat(m_TagAtaData);
```

DecodeRailcarNumber

Decode the railcar number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

unsigned long **DecodeRailcarNumber(const char ****pTagData* **);**

Return Value

This function returns a car number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
int Temp = DecodeRailcarNumber(m_TagAtaData);
```

DecodeRailcarSideIndicator

Decode railcar side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarSideIndicator(const char ****pTagData* **);**

Return Value

This function returns the railcar side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeRailcarSideIndicator(m_TagAtaData);
```

DecodeRailcarNumberOfAxles

Decode the railcar number of axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarNumberOfAxles(const char *** *pTagData* **);**

Return Value

This function returns the railcar number of axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeRailcarNumberOfAxles(m_TagAtaData);
```

DecodeRailcarBearingType

Decode the railcar bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarBearingType(const char ***pTagData* **);**

Return Value

This function returns the railcar bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*buckeye design*)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil filled
6	Reserved
7	Reserved

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeRailcarBearingType(m_TagAtaData);
```

DecodeRailcarLength

Decode the railcar length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarLength(const char ***pTagData* **);**

Return Value

This function returns the railcar length information expressed as an integer in the range of 0 through 4095. The return value represents a length measured in units of decimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The length value represents an AAR Railcar's length expressed in decimeters.

Example

```
Temp = DecodeRailcarLength(m_TagAtaData);
```

DecodeRailcarPlatformIdentifier

Decode the railcar platform identifier information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarPlatformIdentifier(const char ****pTagData* **);**

Return Value

This function returns the railcar platform identifier information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	All equipment except articulated railcars
1	“A” platform
2	“B” platform
3	“C” platform
4	“D” platform
5	“E” platform
6	“F” platform
7	“G” platform
8	“H” platform
9	“I” platform
10	“J” platform
11	“K” platform
12	“L” platform
13	“M” platform
14	“N” platform
15	“O” platform – also applies for platforms beyond the 15th

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeRailcarPlatformIdentifier(m_TagAtaData);
```

DecodeRailcarSpare

Decode the railcar spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeRailcarSpare(const char ***pTagData* **);**

Return Value

This function returns the contents of the railcar *spare* field.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeRailcarSpare(m_TagAtaData);
```

DecodeLocomotiveNumber

Decode the locomotive number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

unsigned long **DecodeLocomotiveNumber(const char ****pTagData* **);**

Return Value

This function returns a locomotive number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeLocomotiveNumber(m_TagAtaData);
```

DecodeLocomotiveSideIndicator

Decode locomotive side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeLocomotiveSideIndicator(const char ***pTagData* **);**

Return Value

This function returns the locomotive side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeLocomotiveSideIndicator(m_TagAtaData);
```

DecodeLocomotiveNumberOfAxles

Decode the locomotive number of axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeLocomotiveNumberOfAxles(const char ****pTagData* **);**

Return Value

This function returns the locomotive number of axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeLocomotiveNumberOfAxles(m_TagAtaData);
```

DecodeLocomotiveBearingType

Decode the locomotive bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeLocomotiveBearingType(const char ****pTagData* **);**

Return Value

This function returns the locomotive bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*Buckeye* design)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil-filled
6	Reserved
7	Reserved

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeLocomotiveBearingType(m_TagAtaData);
```

DecodeLocomotiveLength

Decode the locomotive length field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeLocomotiveLength(const char ****pTagData* **);**

Return Value

This function returns the locomotive length information expressed as an integer in the range of 0 through 510. The return value represents the length in decimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The length value represents an AAR Locomotive's length expressed in decimeters.

Example

```
Temp = DecodeLocomotiveLength(m_TagAtaData);
```

DecodeLocomotiveSpare

Decode the locomotive spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeLocomotiveSpare(const char ***pTagData* **);**

Return Value

This function returns the locomotive spare field information expressed as an integer.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeLocomotiveSpare(m_TagAtaData);
```

DecodeTrailerNumber

Decode a trailer's number information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerNumber(const char ****pTagData***, char ****pTrailerNumber** **);**

Return Value

This function places the decoded trailer number information in the character buffer pointed to by the *pTrailerNumber* parameter.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

pTrailerNumber

Pointer to a buffer that contains the decoded trailer number information when function returns.

Remarks

The DecodeTrailerNumber function converts the data so that the *pTrailerNumber* buffer returns an ASCII nul-terminated string. The buffer for *pTrailerNumber* needs to be allocated a minimum of nine bytes (8 characters + 1 nul).

Example

```
DecodeTrailerNumber(m_TagAtaData, Buffer);
```

DecodeTrailerLength

Decode a trailer's length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerLength(const char ***pTagData* **);**

Return Value

This function returns the trailer length information expressed as an integer in the range of 0 through 2047. The return value represents the length in centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The length value is expressed in centimeters.

Example

```
Temp = DecodeTrailerLength(m_TagAtaData);
```

DecodeTrailerWidth

Decode a trailer's width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerWidth(const char ***pTagData* **);**

Return Value

This function returns the trailer width information expressed as an integer in the range of 0 through 3. The possible return values are defined as follows:

Value	Description
0	Not used
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeTrailerWidth(m_TagAtaData);
```

DecodeTrailerTandemWidth

Decode a trailer's tandem width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerTandemWidth(const char ***pTagData* **);**

Return Value

This function returns the trailer tandem width information expressed as an integer in the range of 0 through 3. The possible return values are defined as follows:

Value	Description
0	Not used
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeTrailerTandemWidth(m_TagAtaData);
```

DecodeTrailerTypeDetail

Decode a trailer's type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerTypeDetail(const char ***pTagData* **);**

Return Value

This function returns the trailer type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Bulk hopper or tank
1	Mechanical refrigerator – underhung
2	General service (non-equipped) dry van
3	Flat bed (including removable sides, platforms, and expandables)
4	Open top
5	Mechanical refrigerator – nose mount
6	Rail compatible trailer, without integral rail wheels
7	Insulated
8	Drop frame (including wedge frames)
9	Special equipped straight floor closed
10	Rail compatible trailer, with integral rail wheels (capable of operation on railroads without an underlying flatcar platform)
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Not used

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeTrailerTypeDetail(m_TagAtaData);
```

DecodeTrailerForwardExtension

Decode a trailer's forward extension information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerForwardExtension(const char *** *pTagData* **);**

Return Value

This function returns the trailer forward extension information expressed as an integer in the range of 30 through 284. The return value represents the length in centimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The forward extension value returned is expressed in units of centimeters.

Example

```
Temp = DecodeTrailerForwardExtension(m_TagAtaData);
```

DecodeTrailerTareWeight

Decode a trailer's tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerTareWeight(const char ***pTagData* **);**

Return Value

This function returns the trailer tare weight information expressed as an integer in the range of 15 through 141. The return value represents the tare weight in units of 100 kilograms.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The *tare weight* value represents an AAR trailer's tare weight expressed in units of 100 kilograms.

Example

```
Temp = DecodeTrailerTareWeight(m_TagAtaData);
```

DecodeTrailerHeight

Decode a trailer's height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeTrailerHeight(const char ***pTagData* **);**

Return Value

This function returns the trailer height information expressed as an integer in the range of 0 through 511. The return value represents the trailer height in units of centimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeTrailerHeight(m_TagAtaData);
```

DecodeChassisNumber

Decode the chassis number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

unsigned long **DecodeChassisNumber(const char ****pTagData* **);**

Return Value

This function returns a chassis number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisNumber(m_TagAtaData);
```

DecodeChassisTypeDetail

Decode chassis type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisTypeDetail(const char ***pTagData* **);**

Return Value

This function returns the chassis type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Extendible
1	Straight
2	Combo
3	Beam slider
4	Rail compatible chassis, with integral rail wheels
5	Rail compatible chassis, without integral rail wheels
6	Fixed length gooseneck
7	Platform
8	Drop frame
9	Tri-purpose
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Others/Not used

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisTypeDetail(m_TagAtaData);
```

DecodeChassisTareWeight

Decode chassis tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisTareWeight(const char ****pTagData* **);**

Return Value

This function returns the chassis tare weight information expressed as an integer in the range of 15 through 77. The return value represents the tare weight in units of 100 kilograms.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The *tare weight* value represents an AAR Chassis' tare weight expressed in units of 100 kilograms. A value of zero (0) indicates that this field is not used.

Example

```
Temp = DecodeChassisTareWeight(m_TagAtaData);
```

DecodeChassisHeight

Decode chassis height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisHeight(const char ***pTagData* **);**

Return Value

This function returns the chassis height information expressed as an integer in the range of 40 through 166. The return value represents the chassis height in units of centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The chassis height value is expressed in units of centimeters.

Example

```
Temp = DecodeChassisHeight(m_TagAtaData);
```

DecodeChassisTandemWidth

Decode chassis tandem width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisTandemWidth(const char ***pTagData* **);**

Return Value

This function returns the chassis tandem width information expressed as an integer in the range of 0 through 3. The possible return values are as follows:

Value	Description
0	Not used/other
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisTandemWidth(m_TagAtaData);
```

DecodeChassisForwardExtension

Decode chassis forward extension information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisForwardExtension(const char ***pTagData* **);**

Return Value

This function returns the chassis forward extension information expressed as an integer in the range of 30 through 154. The return value represents the chassis forward extension in units of centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The chassis forward extension value is expressed in units of centimeters.

Example

```
Temp = DecodeChassisForwardExtension(m_TagAtaData);
```

DecodeChassisKingpinSetting

Decode chassis kingpin setting information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisKingpinSetting(const char ***pTagData* **);**

Return Value

This function returns the chassis kingpin setting information expressed as an integer in the range of 30 through 154. The return value represents the chassis kingpin setting in units of centimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The chassis kingpin setting value is expressed in units of centimeters.

Example

```
Temp = DecodeChassisKingpinSetting(m_TagAtaData);
```

DecodeChassisAxleSpacing

Decode chassis axle spacing information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisAxleSpacing(const char ***pTagData* **);**

Return Value

This function returns the chassis axle spacing information expressed as an integer in the range of 10 through 40. The return value represents the chassis axle spacing in units of decimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The chassis axle spacing value is expressed in units of decimeters.

Example

```
Temp = DecodeChassisAxleSpacing(m_TagAtaData);
```

DecodeChassisRunningGearLocation

Decode chassis running gear location information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisRunningGearLocation(const char ****pTagData* **);**

Return Value

This function returns the chassis running gear location information expressed as an integer in the range of 13 through 43. The return value represents the chassis running gear location in units of decimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

The chassis running gear location value is expressed in units of decimeters.

Example

```
Temp = DecodeChassisRunningGearLocation(m_TagAtaData);
```

DecodeChassisNumberOfLengths

Decode chassis number of lengths information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisNumberOfLengths(const char ****pTagData* **);**

Return Value

This function returns the chassis number of lengths information expressed as an integer in the range of 0 through 7. The possible return values are as follows:

Value	Description
0	Not used
1	1 length
2	2 lengths
3	3 lengths
4	4 lengths
5	5 lengths
6	6 lengths
7	7 or more lengths

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisNumberofLengths(m_TagAtaData);
```

DecodeChassisMinimumLength

Decode chassis minimum length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisMinimumLength(const char ***pTagData* **);**

Return Value

This function returns the chassis minimum length information expressed as an integer in the range of 0 through 2046. The return value represents the chassis minimum length information in units of centimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisMinimumLength(m_TagAtaData);
```

DecodeChassisSpare

Decode chassis spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisSpare(const char ***pTagData* **);**

Return Value

This function returns the value contained in the *spare* data field.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisSpare(m_TagAtaData);
```

DecodeChassisMaximumLength

Decode chassis maximum length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeChassisMaximumLength(const char ***pTagData* **);**

Return Value

This function returns the chassis maximum length information expressed as an integer in the range of 0 through 2046. The return value represents the chassis maximum length information in units of centimeters.

A value of zero (0) indicates that this field is not used.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeChassisMaximumLength(m_TagAtaData);
```

DecodeEndOfTrainNumber

Decode the end-of-train number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

unsigned long **DecodeEndOfTrainNumber(const char ***pTagData* **);**

Return Value

This function returns the end-of-train unit's number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeEndofTrainNumber(m_TagAtaData);
```

DecodeEndOfTrainType

Decode end-of-train type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeEndOfTrainType(const char ****pTagData* **);**

Return Value

This function returns the end-of-train type information expressed as an integer in the range of 0 through 3. The possible return values are as follows:

Value	Description
0	EOT
1	EOT (alternate code use)
2	EOT (alternate code use)
3	Marker light (generally includes brake pressure gauge) - UMLER code 95 - NL/NLU

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeEndofTrainType(m_TagAtaData);
```

DecodeEndOfTrainSideIndicator

Decode end-of-train side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeEndOfTrainSideIndicator(const char *** *pTagData* **);**

Return Value

This function returns the end-of-train side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeEndofTrainSideIndicator(m_TagAtaData);
```

DecodeEndOfTrainSpare1

Decode end-of-train spare1 field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeEndOfTrainSpare1(const char ***pTagData* **);**

Return Value

This function returns the end-of-train spare1 field information expressed as an integer.
Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeEndofTrainSpare1(m_TagAtaData);
```

DecodeEndOfTrainSpare2

Decode end-of-train spare2 field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeEndOfTrainSpare2(const char *** *pTagData* **);**

Return Value

This function returns the end-of-train spare2 field information expressed as an integer.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeEndofTrainSpare2(m_TagAtaData);
```

DecodeIntermodalNumber

Decode the intermodal number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

unsigned long **DecodeIntermodalNumber(const char ***pTagData* **);**

Return Value

This function returns the intermodal number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalNumber(m_TagAtaData);
```

DecodeIntermodalCheckDigit

Decode intermodal check digit information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalCheckDigit(const char ***pTagData* **);**

Return Value

This function returns the intermodal check digit information expressed as an integer in the range of 0 through 9.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalCheckDigit(m_TagAtaData);
```

DecodeIntermodalLength

Decode intermodal length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalLength(const char ***pTagData* **);**

Return Value

This function returns the intermodal length information expressed as an integer in the range of 0 through 2000. The return value represents the length in centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalLength(m_TagAtaData);
```

DecodeIntermodalHeight

Decode intermodal height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalHeight(const char ***pTagData* **);**

Return Value

This function returns the intermodal height information expressed as an integer in the range of 0 through 500. The return value represents the height in centimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalHeight(m_TagAtaData);
```

DecodeIntermodalWidth

Decode intermodal width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalWidth(const char ****pTagData* **);**

Return Value

This function returns the intermodal width information expressed as an integer in the range of 200 through 300. The return value represents the width in centimeters.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalWidth(m_TagAtaData);
```

DecodeIntermodalContainerType

Decode intermodal width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalContainerType(const char ***pTagData* **);**

Return Value

This function returns the intermodal container type information expressed as an integer in the range of 1 through 128. The definition of each possible return value is described in the International Standards Organization Document ISO 6346-1984 (E), Annex G.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalContainerType(m_TagAtaData);
```

DecodeIntermodalMaximumGrossWeight

Decode intermodal maximum gross weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalMaximumGrossWeight(const char *** *pTagData* **);**

Return Value

This function returns the intermodal maximum gross weight information expressed as an integer in the range of 45 through 455. The return value represents the maximum gross weight in units of 100 kilograms.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalMaximumGrossWeight(m_TagAtaData);
```

DecodeIntermodalTareWeight

Decode intermodal tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalTareWeight(const char ****pTagData* **);**

Return Value

This function returns the intermodal car tare weight information expressed as an integer in the range of 0 through 91. The return value represents the tare weight measured in units of 100 kilograms.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalTareWeight(m_TagAtaData);
```

DecodeIntermodalSpare

Decode intermodal spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeIntermodalSpare(const char ***pTagData* **);**

Return Value

This function returns the intermodal spare field information expressed as an integer.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeIntermodalSpare(m_TagAtaData);
```

DecodeMultimodalNumber

Decode the multimodal number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalNumber(const char ***pTagData* **);**

Return Value

This function returns a multimodal number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalNumber(m_TagAtaData);
```

DecodeMultimodalSideIndicator

Decode multimodal side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalSideIndicator(const char ***pTagData* **);**

Return Value

This function returns the multimodal side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodaSideIndicatorr(m_TagAtaData);
```

DecodeMultimodalNumberOfRailAxles

Decode the multimodal number of rail axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalNumberOfRailAxles(const char ***pTagData* **);**

Return Value

This function returns the multimodal number of rail axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalNumberofRailAxles(m_TagAtaData);
```

DecodeMultimodalBearingType

Decode the multimodal bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalBearingType(const char ****pTagData* **);**

Return Value

This function returns the multimodal bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*buckeye design*)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil filled
6	Reserved
7	No rail axle or bearings

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalBearingType(m_TagAtaData);
```

DecodeMultimodalLength

Decode multimodal length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalLength(const char ***pTagData* **);**

Return Value

This function returns the multimodal length information expressed as an integer in the range of 0 through 4095. The return value represents the length in decimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalLength(m_TagAtaData);
```

DecodeMultimodalPlatformIdentifier

Decode the multimodal platform identifier information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalPlatformIdentifier(const char ***pTagData* **);**

Return Value

This function returns the platform identifier information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	All equipment except articulated railcars (includes single platform cars)
1	“A” platform
2	“B” platform
3	“C” platform
4	“D” platform
5	“E” platform
6	“F” platform
7	“G” platform
8	“H” platform
9	“I” platform
10	“J” platform
11	“K” platform
12	“L” platform
13	“M” platform
14	“N” platform
15	“O” platform – also applies for platforms beyond the 15th

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalPlatformIdentifier(m_TagAtaData);
```

DecodeMultimodalTypeDetail

Decode the multimodal type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int DecodeMultimodalTypeDetail(const char ***pTagData* **);

Return Value

This function returns the multimodal type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Data not provided
1	Adapter car (stand-alone vehicle to connect road railer to conventional equipment)
2	Transition rail truck (e.g., coupler mate – rail truck used to connect road railer to conventional equipment)
3	Rail truck (bogie)
4	Rail compatible trailer, with integral rail wheels (e.g., Road railer Mark IV)
5	Rail compatible trailer, without integral rail wheels (e.g. Road railer Mark IV)
6	Bi-modal maintenance-of-way equipment
7	Reserved
8	Reserved
9	Reserved
10	Iron highway platform unit
11	Iron highway power unit
12	Reserved
13	Reserved
14	Reserved
15	Reserved

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalTypeDetail(m_TagAtaData);
```

DecodeMultimodalSpare

Decode multimodal spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

int **DecodeMultimodalSpare(const char ***pTagData* **);**

Return Value

This function returns the multimodal spare field information expressed as an integer.

Parameters

pTagData

Pointer to a buffer containing the data to be decoded.

Remarks

none

Example

```
Temp = DecodeMultimodalSpare(m_TagAtaData);
```

IP3UTILL DLL Methods

This section describes the IP3UTILS DLL Methods. Each method's section lists the code, the value returned, any parameters, any remarks, and an example of the code.

IP3Scanner

Constructor

Public IP3ReaderOpen( );

Return Value

none

Parameters

none

Remarks

none

Example

```
IP3Scanner myScanner = new IP3Scanner();
```

IP3Shutdown

Close an open Encompass 1 reader connection.

Public void **IP3Shutdown(void);**

Return Value

Parameters

none

Remarks

Closes the Encompass 1 reader and turns off the hardware events;

Example

```
myScanner.Shutdown()
```

Identify

Use an open Encompass 1 reader connection to command the reader device to attempt to identify a tag if present.

public int Identify(out string TagId)

Return Value

A return value of eTT_None indicates that no tag data was read.

A return value of eTT_Ego indicates that an eGo tag was identified and the tag contains no specially formatted data. In this case, *pTagData* contains the eGo tag's eight byte serial number.

A return value of eTT_ATA_HALF indicates that *pTagData* contains Wiegand-formatted data.

A return value of eTT_ATA indicates that *pTagData* contains ATA-formatted data.

A return value of eTT_Error indicates that a communications error occurred.

Parameters

TagID

String where the function places the tag data on success.

Remarks

A return value of eTT_Error indicates that a communications error occurred. If this happens, close and then re-open the connection to the reader before trying to perform any further identify or read operations.

Example

```
int RC = myScanner.Identify(out m_TagId);
```

ReadEgoAta

Use an open Encompass 1 reader connection to read the ATA data frame from an eGo tag.

public int ReadEgoAta(ref string TagId, out string AtaData)

Return Value

Positive return value indicates success.

A zero return value indicates a read error.

A negative return value indicates a communications error occurred.

Parameters

`TagId`

String containing the tag data that was returned from a previous call to the *IP3Identify()* function in which the function return code indicated that the data read was eGo Tag ID data.

`AtaData`

String where the function places the tag data on success.

Remarks

A negative return value indicates that a communications error occurred. If this happens, close and then re-open the connection to the reader before trying to perform any further identify or read operations.

Example

```
int RC = myScanner.ReadEgoAta(ref m_TagId,out m_TagAtaData);
```

IP3DecodeATAData

Decode the 20 ATA six-bit ASCII characters contained in the ATA-formatted data specified by the *pTagData* parameter.

public void IP3DecodeAtaData(string pTagData,

out string pAtaData, int FrameSize)

Return Value

This function returns 20 eight-bit ASCII characters, along with a terminating nul character, in the buffer space pointed to by the *pATACharacters* parameter.

Parameters

pTagData

String containing the data to be decoded.

pATACharacters

String that contains the ATA characters when the function returns.

Remarks

This function should only be used on ATA tag data that is not AAR-formatted data.

Example

```
IP3DecodeAtaData(m_TagId, out Buffer, FrameSize);
```

IP3DecodeFirstCheckSum

Decode the value of the first checksum field contained in the ATA-formatted data specified by the *pTagData* parameter.

public int IP3DecodeFirstCheckSum(string pTagData)

Return Value

This function returns the value of the first checksum field. Valid values are 0, 1, 2, or 3.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int temp = IP3DecodeFirstCheckSum(m_TagAtaData);
```

IP3DecodeHalfFrameMarker

Decode the value of the first frame marker field (also called the half frame marker field) contained in the ATA-formatted data specified by the *pTagData* parameter.

public int IP3DecodeHalfFrameMarker(string pTagData)

Return Value

This function returns the value of the first frame marker field. A value of 2 indicates that only the first half ATA frame contains usable data. A value of 4 indicates that both the first and second half ATA frames contain usable data.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int temp = IP3DecodeHalfFrameMarker(m_TagAtaData);
```

IP3DecodeSecondCheckSum

Decode the value of the second checksum field contained in the ATA-formatted data specified by the *pTagData* parameter.

public int IP3DecodeSecondChecksum(string pTagData)

Return Value

This function returns the value of the second checksum field. Valid values are 0, 1, 2, or 3.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int temp = myScanner.IP3DecodeSecondCheckSum(m_TagAtaData);
```

IP3DecodeFullFrameMarker

Decode the value of the second frame marker field (also called the full frame marker field) contained in the ATA-formatted data specified by the *pTagData* parameter.

```
public int IP3DecodeFullFramerMarker(string pTagData)
```

Return Value

This function returns the value of the second frame marker field.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int temp = myScanner.IP3DecodeFullFramMarker(m_TagAtaData);
```

IP3IsAAR

Determine if data contained in a buffer is AAR-formatted data.

public int IP3IsAAR(string pTagData)

Return Value

This function returns a value of 1 if the data contained in *pTagData* is AAR-formatted data, otherwise, the function returns a value of 0.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

int iAAR = `myScanner.IP3IsAAR(m_TagAtaData);`

IP3DecodeEquipmentGroup

Decode the equipment group code value contained in the AAR-formatted data block provided in the *pTagData* parameter.

public int IP3DecodeEquipmentGroup(string pTagData)

Return Value

This function returns the value contained in the equipment group code field. Valid range of values is 0-31. For freight and passenger railcars, this value is decimal 19.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

int groupCode = `myScanner.IP3`DecodeEquipmentGroup(m_TagAtaData);

IP3DecodeTagType

Decode the tag type value contained in the AAR-formatted data block provided in the *pTagData* parameter.

public int IP3DecodeTagType(string pTagData)

Return Value

This function returns the value contained in the tag type field. This value always equals decimal two (2). If this value is equal to anything other than decimal two, do not use these functions for any further decoding of the data contained in *pTagData*.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int tagType = myScanner.IP3DecodeTagType(m_TagAtaData);
```

IP3DecodeEquipmentInitial

Decode the equipment initial value contained in the AAR-formatted data block provided in the *pTagData* parameter.

```
public void IP3DecodeEquipmentInitial(string pTagData,out
string pInititals)
```

Return Value

This function decodes the four-character equipment initial data contained in the AAR-formatted data provided by the *pTagData* parameter and returns the equipment initial characters in the *pEquipmentInitials* parameter.

Parameters

pTagData

String containing the data to be decoded.

pEquipmentInitials

String that contains the Equipment Initial characters on successful function completion.

Remarks

none

Example

```
myScanner.IP3DecodeEquipmentInitial(m_TagAtaData, Buffer);
```

IP3DecodeReserved

Decode the Reserved value contained in the AAR-formatted data block provided in the *pTagData* parameter.

public int IP3DecodeReserved(string pTagData)

Return Value

This function returns the reserved field of an AAR-formatted tag.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
myScanner.IP3DecodeReserved(m_TagAtaData);
```

IP3DecodeSecurityBits

Decode the security bits information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeSecurityBits(string pTagData)

Return Value

This function returns the security bits in an AAR-formatted tag.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
myScanner.IP3DecodeSecurityBits(m_TagAtaData);
```

IP3DecodeSecurity

Decode the security bits information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public void DecodeSecurity(string pTagData,

out string pSecurityChars)

Return Value

This function decodes the four-character security data contained in the AAR-formatted data provided by the *pTagData* parameter and returns the security characters in the *pSecurityChars* parameter.

Parameters

pTagData

String containing the data to be decoded.

pSecurityChars

String that contains the two security characters (plus a nul character) when the function returns.

Remarks

none

Example

```
myScanner.IP3DecodeSecurity(m_TagAtaData, out Buffer);
```

IP3DecodeDataFormat

Decode the data format information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeDataFormat(string pTagData)

Return Value

This function returns the data format of an AAR-formatted tag.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int DataFormatCode = myScanner.IP3DecodeDataFormat(m_TagAtaData);
```

IP3DecodeRailcarNumber

Decode the railcar number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeRailcarNumber(string pTagData)

Return Value

This function returns a car number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int Temp = myScanner.IP3DecodeRailcarNumber(m_TagAtaData);
```

IP3DecodeRailcarSideIndicator

Decode railcar side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeRailcarSideIndicator(string pTagData)

Return Value

This function returns the railcar side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
int car = myScanner.IP3DecodeRailcarSideIndicator(m_TagAtaData);
```

IP3DecodeRailcarNumberOfAxles

Decode the railcar number of axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeRailcarNumberOfAxles(string pTagData)

Return Value

This function returns the railcar number of axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeRailcarNumberOfAxles(m_TagAtaData);
```

IP3DecodeRailcarBearingType

Decode the railcar bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeRailcarBearingType(string pTagData)

Return Value

This function returns the railcar bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*buckeye design*)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil filled
6	Reserved
7	Reserved

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeRailcarBearingType(m_TagAtaData);
```

IP3DecodeRailcarLength

Decode the railcar length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeRailcarLength(string pTagData)

Return Value

This function returns the railcar length information expressed as an integer in the range of 0 through 4095. The return value represents a length measured in units of decimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The length value represents an AAR Railcar's length expressed in decimeters.

Example

```
Temp = myScanner.IP3DecodeRailcarLength(m_TagAtaData);
```

IP3DecodeRailcarPlatformIdentifier

Decode the railcar platform identifier information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeRailcarPlatformIdentifier(string pTagData)
```

Return Value

This function returns the railcar platform identifier information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	All equipment except articulated railcars
1	“A” platform
2	“B” platform
3	“C” platform
4	“D” platform
5	“E” platform
6	“F” platform
7	“G” platform
8	“H” platform
9	“I” platform
10	“J” platform
11	“K” platform
12	“L” platform
13	“M” platform
14	“N” platform
15	“O” platform – also applies for platforms beyond the 15th

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScaner.IP3DecodeRailcarPlatformIdentifier(m_TagAtaData);
```

IP3DecodeRailcarSpare

Decode the railcar spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeRailcarSpare(string pTagData)
```

Return Value

This function returns the contents of the railcar *spare* field.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeRailcarSpare(m_TagAtaData);
```

IP3DecodeLocomotiveNumber

Decode the locomotive number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveNumber(string pTagData)
```

Return Value

This function returns a locomotive number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeLocomotiveNumber(m_TagAtaData);
```

IP3DecodeLocomotiveSideIndicator

Decode locomotive side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveSideIndicator(string pTagData)
```

Return Value

This function returns the locomotive side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeLocomotiveSideIndicator(m_TagAtaData);
```

IP3DecodeLocomotiveNumberOfAxles

Decode the locomotive number of axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveNumberOfAxles(string pTagData)
```

Return Value

This function returns the locomotive number of axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeLocomotiveNumberOfAxles(m_TagAtaData);
```

IP3DecodeLocomotiveBearingType

Decode the locomotive bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveBearingType(string pTagData)
```

Return Value

This function returns the locomotive bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*Buckeye* design)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil-filled
6	Reserved
7	Reserved

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeLocomotiveBearingType(m_TagAtaData);
```

IP3DecodeLocomotiveLength

Decode the locomotive length field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveLength(string pTagData)
```

Return Value

This function returns the locomotive length information expressed as an integer in the range of 0 through 510. The return value represents the length in decimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The length value represents an AAR Locomotive's length expressed in decimeters.

Example

```
Temp = myScanner.IP3DecodeLocomotiveLength(m_TagAtaData);
```

IP3DecodeLocomotiveSpare

Decode the locomotive spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeLocomotiveSpare(string pTagData)
```

Return Value

This function returns the locomotive spare field information expressed as an integer.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeLocomotiveSpare(m_TagAtaData);
```

IP3DecodeTrailerNumber

Decode a trailer's number information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeTrailerNumber(string pTagData, out string pTrailerNumber)

Return Value

This function places the decoded trailer number information in the character buffer pointed to by the *pTrailerNumber* parameter.

Parameters

pTagData

String containing the data to be decoded.

pTrailerNumber

String that contains the decoded trailer number information when function returns.

Remarks

The DecodeTrailerNumber function converts the data so that the *pTrailerNumber* buffer returns an ASCII nul-terminated string. The buffer for *pTrailerNumber* needs to be allocated a minimum of nine bytes (8 characters + 1 nul).

Example

```
myScanner.IP3DecodeTrailerNumber(m_TagAtaData, out Buffer);
```

IP3DecodeTrailerLength

Decode a trailer's length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeTrailerLength(string pTagData)

Return Value

This function returns the trailer length information expressed as an integer in the range of 0 through 2047. The return value represents the length in centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The length value is expressed in centimeters.

Example

```
Temp = myScanner.IP3DecodeTrailerLength(m_TagAtaData);
```

IP3DecodeTrailerWidth

Decode a trailer's width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerWidth(string pTagData)
```

Return Value

This function returns the trailer width information expressed as an integer in the range of 0 through 3. The possible return values are defined as follows:

Value	Description
0	Not used
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeTrailerWidth(m_TagAtaData);
```

IP3DecodeTrailerTandemWidth

Decode a trailer's tandem width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerTandemWidth(string pTagData)
```

Return Value

This function returns the trailer tandem width information expressed as an integer in the range of 0 through 3. The possible return values are defined as follows:

Value	Description
0	Not used
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeTrailerTandemWidth(m_TagAtaData);
```

IP3DecodeTrailerTypeDetail

Decode a trailer's type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerTypeDetail(string pTagData)
```

Return Value

This function returns the trailer type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Bulk hopper or tank
1	Mechanical refrigerator – underhung
2	General service (non-equipped) dry van
3	Flat bed (including removable sides, platforms, and expandables)
4	Open top
5	Mechanical refrigerator – nose mount
6	Rail compatible trailer, without integral rail wheels
7	Insulated
8	Drop frame (including wedge frames)
9	Special equipped straight floor closed
10	Rail compatible trailer, with integral rail wheels (capable of operation on railroads without an underlying flatcar platform)
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Not used

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeTrailerTypeDetail(m_TagAtaData);
```

IP3DecodeTrailerForwardExtension

Decode a trailer's forward extension information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerForwardExtension(string pTagData)
```

Return Value

This function returns the trailer forward extension information expressed as an integer in the range of 30 through 284. The return value represents the length in centimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The forward extension value returned is expressed in units of centimeters.

Example

```
Temp = myScanner.IP3DecodeTrailerForwardExtension(m_TagAtaData);
```

IP3DecodeTrailerTareWeight

Decode a trailer's tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerTareWeight(string pTagData)
```

Return Value

This function returns the trailer tare weight information expressed as an integer in the range of 15 through 141. The return value represents the tare weight in units of 100 kilograms.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The *tare weight* value represents an AAR trailer's tare weight expressed in units of 100 kilograms.

Example

```
Temp = myScanner.IP3DecodeTrailerTareWeight(m_TagAtaData);
```

IP3DecodeTrailerHeight

Decode a trailer's height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeTrailerHeight(string pTagData)
```

Return Value

This function returns the trailer height information expressed as an integer in the range of 0 through 511. The return value represents the trailer height in units of centimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeTrailerHeight(m_TagAtaData);
```

IP3DecodeChassisNumber

Decode the chassis number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisNumber(string pTagData)
```

Return Value

This function returns a chassis number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisNumber(m_TagAtaData);
```

IP3DecodeChassisTypeDetail

Decode chassis type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisTypeDetail(string pTagData)
```

Return Value

This function returns the chassis type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Extendible
1	Straight
2	Combo
3	Beam slider
4	Rail compatible chassis, with integral rail wheels
5	Rail compatible chassis, without integral rail wheels
6	Fixed length gooseneck
7	Platform
8	Drop frame
9	Tri-purpose
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Others/Not used

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisTypeDetail(m_TagAtaData);
```

IP3DecodeChassisTareWeight

Decode chassis tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisTareWeight(string pTagData)
```

Return Value

This function returns the chassis tare weight information expressed as an integer in the range of 15 through 77. The return value represents the tare weight in units of 100 kilograms.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The *tare weight* value represents an AAR Chassis' tare weight expressed in units of 100 kilograms. A value of zero (0) indicates that this field is not used.

Example

```
Temp = myScanner.IP3DecodeChassisTareWeight(m_TagAtaData);
```

IP3DecodeChassisHeight

Decode chassis height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisHeight(string pTagData)
```

Return Value

This function returns the chassis height information expressed as an integer in the range of 40 through 166. The return value represents the chassis height in units of centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The chassis height value is expressed in units of centimeters.

Example

```
Temp = myScanner.IP3DecodeChassisHeight(m_TagAtaData);
```

IP3DecodeChassisTandemWidth

Decode chassis tandem width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisTandemWidth(string pTagData)
```

Return Value

This function returns the chassis tandem width information expressed as an integer in the range of 0 through 3. The possible return values are as follows:

Value	Description
0	Not used/other
1	96 in./2.5 m or less
2	More than 96 in./2.5 m but not more than 102 in./2.6 m
3	More than 102 in./2.6 m

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisTandemWidth(m_TagAtaData);
```

IP3DecodeChassisForwardExtension

Decode chassis forward extension information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisForwardExtension(string pTagData)
```

Return Value

This function returns the chassis forward extension information expressed as an integer in the range of 30 through 154. The return value represents the chassis forward extension in units of centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The chassis forward extension value is expressed in units of centimeters.

Example

```
Temp = myScanner.IP3DecodeChassisForwardExtension(m_TagAtaData);
```

IP3DecodeChassisKingpinSetting

Decode chassis kingpin setting information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisKingpinSetting(string pTagData)
```

Return Value

This function returns the chassis kingpin setting information expressed as an integer in the range of 30 through 154. The return value represents the chassis kingpin setting in units of centimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The chassis kingpin setting value is expressed in units of centimeters.

Example

```
Temp = myScanner.IP3DecodeChassisKingpinSetting(m_TagAtaData);
```

IP3DecodeChassisAxleSpacing

Decode chassis axle spacing information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisAxleSpacing(string pTagData)
```

Return Value

This function returns the chassis axle spacing information expressed as an integer in the range of 10 through 40. The return value represents the chassis axle spacing in units of decimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The chassis axle spacing value is expressed in units of decimeters.

Example

```
Temp = myScanner.IP3DecodeChassisAxleSpacing(m_TagAtaData);
```

IP3DecodeChassisRunningGearLocation

Decode chassis running gear location information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisRunningGearLocation(string pTagData)
```

Return Value

This function returns the chassis running gear location information expressed as an integer in the range of 13 through 43. The return value represents the chassis running gear location in units of decimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

String containing the data to be decoded.

Remarks

The chassis running gear location value is expressed in units of decimeters.

Example

```
Temp = myScanner.IP3DecodeChassisRunningGearLocation(m_TagAtaData);
```

IP3DecodeChassisNumberOfLengths

Decode chassis number of lengths information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisNumberOfLengths(string pTagData)
```

Return Value

This function returns the chassis number of lengths information expressed as an integer in the range of 0 through 7. The possible return values are as follows:

Value	Description
0	Not used
1	1 length
2	2 lengths
3	3 lengths
4	4 lengths
5	5 lengths
6	6 lengths
7	7 or more lengths

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisNumberofLengths(m_TagAtaData);
```

IP3DecodeChassisMinimumLength

Decode chassis minimum length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisMinimumLength(string pTagData)
```

Return Value

This function returns the chassis minimum length information expressed as an integer in the range of 0 through 2046. The return value represents the chassis minimum length information in units of centimeters.

A value of zero (0) indicates that this field does not apply or the value is unknown.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisMinimumLength(m_TagAtaData);
```

IP3DecodeChassisSpare

Decode chassis spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisSpare(string pTagData)
```

Return Value

This function returns the value contained in the *spare* data field.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisSpare(m_TagAtaData);
```

IP3DecodeChassisMaximumLength

Decode chassis maximum length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeChassisMaximumLength(string pTagData)
```

Return Value

This function returns the chassis maximum length information expressed as an integer in the range of 0 through 2046. The return value represents the chassis maximum length information in units of centimeters.

A value of zero (0) indicates that this field is not used.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeChassisMaximumLength(m_TagAtaData);
```

IP3DecodeEndOfTrainNumber

Decode the end-of-train number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeEndOfTrainNumber(string pTagData)

Return Value

This function returns the end-of-train unit's number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeEndofTrainNumber(m_TagAtaData);
```

IP3DecodeEndOfTrainType

Decode end-of-train type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeEndOfTrainType(string pTagData)
```

Return Value

This function returns the end-of-train type information expressed as an integer in the range of 0 through 3. The possible return values are as follows:

Value	Description
0	EOT
1	EOT (alternate code use)
2	EOT (alternate code use)
3	Marker light (generally includes brake pressure gauge) - UMLER code 95 - NL/NLU

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeEndofTrainType(m_TagAtaData);
```

IP3DecodeEndOfTrainSideIndicator

Decode end-of-train side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeEndOfTrainSideIndicator(string pTagData)
```

Return Value

This function returns the end-of-train side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeEndofTrainSideIndicator(m_TagAtaData);
```

IP3DecodeEndOfTrainSpare1

Decode end-of-train spare1 field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeEndOfTrainSpare1(string pTagData)
```

Return Value

This function returns the end-of-train spare1 field information expressed as an integer.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeEndofTrainSpare1(m_TagAtaData);
```

IP3DecodeEndOfTrainSpare2

Decode end-of-train spare2 field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeEndOfTrainSpare2(string pTagData)
```

Return Value

This function returns the end-of-train spare2 field information expressed as an integer.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeEndofTrainSpare2(m_TagAtaData);
```

IP3DecodeIntermodalNumber

Decode the intermodal number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalNumber(string pTagData)
```

Return Value

This function returns the intermodal number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalNumber(m_TagAtaData);
```

IP3DecodeIntermodalCheckDigit

Decode intermodal check digit information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalCheckDigit(string pTagData)
```

Return Value

This function returns the intermodal check digit information expressed as an integer in the range of 0 through 9.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalCheckDigit(m_TagAtaData);
```

IP3DecodeIntermodalLength

Decode intermodal length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalLength(string pTagData)
```

Return Value

This function returns the intermodal length information expressed as an integer in the range of 0 through 2000. The return value represents the length in centimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalLength(m_TagAtaData);
```

IP3DecodeIntermodalHeight

Decode intermodal height information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalHeight(string pTagData)
```

Return Value

This function returns the intermodal height information expressed as an integer in the range of 0 through 500. The return value represents the height in centimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalHeight(m_TagAtaData);
```

IP3DecodeIntermodalWidth

Decode intermodal width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalWidth(string pTagData)
```

Return Value

This function returns the intermodal width information expressed as an integer in the range of 200 through 300. The return value represents the width in centimeters.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalWidth(m_TagAtaData);
```

IP3DecodeIntermodalContainerType

Decode intermodal width information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalContainerType(string pTagData)
```

Return Value

This function returns the intermodal container type information expressed as an integer in the range of 1 through 128. The definition of each possible return value is described in the International Standards Organization Document ISO 6346-1984 (E), Annex G.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalContainerType(m_TagAtaData);
```

IP3DecodeIntermodalMaximumGrossWeight

Decode intermodal maximum gross weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalMaximumGrossWeight(string pTag-
Data)
```

Return Value

This function returns the intermodal maximum gross weight information expressed as an integer in the range of 45 through 455. The return value represents the maximum gross weight in units of 100 kilograms.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalMaximumGrossWeight(m_TagAtaData);
```

IP3DecodeIntermodalTareWeight

Decode intermodal tare weight information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalTareWeight(string pTagData)
```

Return Value

This function returns the intermodal car tare weight information expressed as an integer in the range of 0 through 91. The return value represents the tare weight measured in units of 100 kilograms.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalTareWeight(m_TagAtaData);
```

IP3DecodeIntermodalSpare

Decode intermodal spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeIntermodalSpare(string pTagData)
```

Return Value

This function returns the intermodal spare field information expressed as an integer.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeIntermodalSpare(m_TagAtaData);
```

IP3DecodeMultimodalNumber

Decode the multimodal number value contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

public int IP3DecodeMultimodalNumber(string pTagData)

Return Value

This function returns a multimodal number expressed as an integer in the range of 0 through 999999.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalNumber(m_TagAtaData);
```

IP3DecodeMultimodalSideIndicator

Decode multimodal side indicator information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalSideIndicator(string pTagData)
```

Return Value

This function returns the multimodal side indicator information expressed as an integer in the range of 0 through 1. The possible return values are as follows:

Value	Description
0	Left side
1	Right side

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodaSideIndicatorr(m_TagAtaData);
```

IP3DecodeMultimodalNumberOfRailAxles

Decode the multimodal number of rail axles information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalNumberOfRailAxles(string pTag-
Data)
```

Return Value

This function returns the multimodal number of rail axles expressed as an integer in the range of 1 through 32.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalNumberofRailAxles(m_TagAtaData);
```

IP3DecodeMultimodalBearingType

Decode the multimodal bearing type information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalBearingType(string pTagData)
```

Return Value

This function returns the multimodal bearing type information expressed as an integer in the range of 0 through 7. The possible return values are defined as follows:

Value	Description
0	Plain bearings
1	Roller bearings, not otherwise classified
2	Roller bearings, inboard
3	Roller bearings, 3-axle truck, 1-axle obstructed (*buckeye design*)
4	Roller bearings, plain bearing housing
5	Roller bearings, cylindrical oil filled
6	Reserved
7	No rail axle or bearings

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalBearingType(m_TagAtaData);
```

IP3DecodeMultimodalLength

Decode multimodal length information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalLength(string pTagData)
```

Return Value

This function returns the multimodal length information expressed as an integer in the range of 0 through 4095. The return value represents the length in decimeters.

A value of zero (0) indicates that this field does not apply.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalLength(m_TagAtaData);
```

IP3DecodeMultimodalPlatformIdentifier

Decode the multimodal platform identifier information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalPlatformIdentifier(string pTag-
Data)
```

Return Value

This function returns the platform identifier information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	All equipment except articulated railcars (includes single platform cars)
1	“A” platform
2	“B” platform
3	“C” platform
4	“D” platform
5	“E” platform
6	“F” platform
7	“G” platform
8	“H” platform
9	“I” platform
10	“J” platform
11	“K” platform
12	“L” platform
13	“M” platform
14	“N” platform
15	“O” platform – also applies for platforms beyond the 15th

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalPlatformIdentifier(m_TagAtaData);
```

IP3DecodeMultimodalTypeDetail

Decode the multimodal type detail information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalTypeDetail(string pTagData)
```

Return Value

This function returns the multimodal type detail information expressed as an integer in the range of 0 through 15. The possible return values are defined as follows:

Value	Description
0	Data not provided
1	Adapter car (stand-alone vehicle to connect road railer to conventional equipment)
2	Transition rail truck (e.g., coupler mate – rail truck used to connect road railer to conventional equipment)
3	Rail truck (bogie)
4	Rail compatible trailer, with integral rail wheels (e.g., Road railer Mark IV)
5	Rail compatible trailer, without integral rail wheels (e.g. Road railer Mark IV)
6	Bimodal maintenance-of-way equipment
7	Reserved
8	Reserved
9	Reserved
10	Iron highway platform unit
11	Iron highway power unit
12	Reserved
13	Reserved
14	Reserved
15	Reserved

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalTypeDetail(m_TagAtaData);
```

IP3DecodeMultimodalSpare

Decode multimodal spare field information contained in the AAR-formatted data block pointed to by the *pTagData* parameter.

```
public int IP3DecodeMultimodalSpare(string pTagData)
```

Return Value

This function returns the multimodal spare field information expressed as an integer.

Parameters

pTagData

String containing the data to be decoded.

Remarks

none

Example

```
Temp = myScanner.IP3DecodeMultimodalSpare(m_TagAtaData);
```

A

Encompass 1 Scan Handle Specifications

Appendix A

Encompass 1 RFID Scan Handle Specifications

This appendix lists the Encompass® 1 product specifications.

Encompass 1 RFID Scan Handle Specifications

This section lists the Encompass 1 RFID Scan Handle performance and product specifications. See the *Intermec 700 Mobile Computer Quick Start Guide* for performance and product specifications for that device.

Product Performance

Using the scan handle continuously to read tags may cause the unit to overheat. Overheating causes the scan handle to stop reading tags. If this happens, allow the scan handle to cool to a temperature range of -4°F to +131°F (-20°C to +55°C) before reading more tags.

Encompass 1 RFID Scan Handle Specifications

Frequency

Table A-1 Frequency Specifications

Frequency Reference Source	**Specification Detail**
Source type	Frequency hopper
Frequency band	902 to 928 MHz
Transmitter	
Output power (1 watt maximum)	Minimum: 28.5 dBm Typical: 29.5 dBm Maximum: 30.0 dBm
Modulation	99%, 40 dB on/off
Data rate	eGo® tags: 33 to 40 kbps ATA tags: 10 kbps

Performance

Table A-2 Overall Performance Specifications

Dispatch Rates	Specification Detail
RFID tag ID rate	Read at least 6 eGo tags per second at a distance of 3 feet (0.9 meters).
RFID tag read distance	ATA tags: 4 feet (1.2 meters) eGo tags: 3 feet (0.9 meters)
RFID tag data exchange rates	Read an eGo tag containing 8 bytes of data within 50 milliseconds (ms). Write a single byte of data to an eGo tag at an average rate of 75 ms at a distance of 2.1 feet (0.6 m).

Environmental

Table A-3 Environmental Specifications

Temperature Ranges	Specification Detail
Operating	-4°F to +131°F (-20°C to +55°C)
Storage	-40°F to +158°F (-40°C to +70°C)
Humidity	
Operating	0 to 95% relative, non-condensing

Firmware Architecture

Table A-4 Firmware Architecture Specifications

Firmware	Specification Detail
Protocol/compatibility	Communicates in one mode: application peripheral interface (API). ANSI NCITS 256.2000 for API, part 2, part 3-1

Safety/Regulatory/Compliance

Table A-5 Safety/Regulatory/Compliance Specifications

Safety and Regulatory Approvals	**Specification Detail**
Scan handle	cULus listed accessory CB report for international product safety FCC OET Bulletin 65, Evaluating Compliance with FCC Guidelines for Human Exposure to Radio Frequency Electromagnetic Fields, for general population uncontrolled exposure when installed in accordance with TransCore-approved antennas
Electromagnetic Compatibility	
Scan handle	FCC Part 15/Industry Canada ICES-003 Class B digital emissions