Table of Contents

1. Overview 5
2. Development Tools 5
3. Development Environment Changes 5
3.1. File Naming Conventions 5
3.2. Worldgroup Development Environment Requirements 5
3.3. Environment Variables 5
3.4. Development/Runtime Tree 6
3.5. The \WGDEV\SRC directory 6
3.6. PATH Statement 6
3.7. Files contained in each project 6
3.7.1. #ifdef'ing files 6
3.7.2. Making Worldgroup 3.00 .MAK files work for 3.12 7
3.7.3. New .MAK file macros 9
3.7.4. The $(GCBUILD) macro and GCBUILD directories 9
3.7.5. Creating .VIR files from within .MAK files. 10
3.8. Compilation Batch Files 10
3.8.1. MKNT.BAT and MKNTD.BAT 11
3.8.2. MKDOSL.BAT and MKDOSLD.BAT 11
3.8.3. MKDOSP.BAT and MKDOSPD.BAT 11
3.8.4. MK31V.BAT and MK31VD.BAT 11
3.8.5. MK31W.BAT and MK31VW.BAT 11
3.8.6. MKMVC.BAT 11
3.8.7. MKVB.BAT and MKVBC.BAT 11
3.8.8. MAKEWGS.BAT 12
3.9. New Development Utilites 12
3.9.1. GALCOPY.EXE 12
3.9.2. GALMERGE.EXE 12
3.9.3. GALMKDIR.EXE 12
3.9.4. GALPARSE.EXE 12
3.9.5. GALSYNC.EXE 13
3.9.6. GALPROD.EXE 13
4. Data File Conversion 14
4.1. Backward compatibility 14
4.2. WGSDFCVT.EXE 14
4.3. Conversion file theory 14
4.4. .CVT file formats 15
4.4.1. The ; operator 15
4.4.2. Description: 15
4.4.3. Developer: 16
4.4.4. CustomConversion: 16
4.4.5. BeginFile: and EndFile 16
4.4.6. Data Fields 16
4.4.7. TYPE=STRUCT and EndStruct 17
4.5. New Data File Conversion API 17
4.5.1. DFCLIST structure 17
4.5.2. DFCFILE structure 18
4.5.3. DFCDATAFIELDS structure 18
4.5.4. DFCOFFSETS structure 19
4.5.5. DFCFILEINFO structure 20
4.5.6. DFCPFCONVERT structure 21
4.5.7. dfcAskBackup() 21
4.5.8. dfcBackupDirectory() 21
4.5.9. dfcBackupRequired() 21
4.5.10. dfcConvertDefault() 21

4.5.11. dfcConvertPFDefault() 22
4.5.12. dfcDisplayBackup() 22
4.5.13. dfcDisplayInfo() 22
4.5.14. dfcDisplayStats() 23
4.5.15. dfcDisplayShutdown() 23
4.5.16. dfcError() 23
4.5.17. dfcFieldSize() 24
4.5.18. dfcFilePlatform() 24
4.5.19. dfcFindOffset() 24
4.5.20. dfcGetFDA() 25
4.5.21. dfcRegister() 25
4.5.22. dfcRegisterGDB() 26
4.5.23. DFCCONVERT (Conversion functions) 26
4.6. Creating a custom conversion program 27
4.6.1. Create the DFCLIST structure 27
4.6.2. Create the DFCFILE structures 27
4.6.3. Create the DFCDATAFIELDS structures 27
4.6.4. Creating the platform conversion function list 28
4.6.5. Create the custom conversion functions 28
4.6.6. Putting it all together 29
4.6.7. Adding another data file version 29
5. InstallShield V5.1 31
5.1. New Features 31
5.1.1. Uninstall 31
5.1.2. Registry 31
5.1.3. Media 31
5.1.4. Modifiable MSG Files 31
5.1.5. Run Executable 31
5.1.6. Activation Codes 31
5.1.7. Folders 31
5.1.8. INI file 32
5.2. Requirements 32
5.3. Example 32
- WelcomeBmp - BMP file to show 38
6. Year 2000 issues 39
6.1. Worldgroup Baseline Changes Required For Year 2000 Compliance 39
6.1.1. User Account Birth Date 39
6.1.2. Audit Trail 39
6.1.3. Client Application Date Entry 39
6.2. API Changes And Additions 40
6.2.1. Birth Date Manipulation Functions 40
6.2.2. Date and Time API 40
6.2.3. Audit Trail API 41
6.3. Compliance Guidelines 43
6.3.1. Handling Birth Date 43
6.3.2. Using the New Audit Trail 43
6.3.3. Date Output 43
6.3.4. Date Input 44
7. Searchable Memory Blocks 45
7.1. SMB API 45
7.1.1. SMBKEYTABLE structure 45
7.1.2. smbClose() 45
7.1.3. smbCurrentData() 45
7.1.4. smbCurrentNumber() 45
7.1.5. smbDelete() 46
7.1.6. smbGetBynum() 46

7.1.7. smbGetEqual() 46
7.1.8. smbGetGreater() and smbGetLess() 47
7.1.9. smbGetHigh() and smbGetLow() 47
7.1.10. smbGetPrevious() and smbGetNext() 48
7.1.11. smbInit() 48
7.1.12. smbInsert() 48
7.1.13. smbOpen() 48
7.1.14. The Query Routines 49
7.1.15. The default compare functions 49
7.2. Compare functions 49
7.3. Sample SMB application 50
8. Audit Trail API 52
8.1. OVERVIEW 52
8.1.1. File Format 52
8.1.2. Online Context 52
8.1.3. Example 53
8.2. Result Codes 55
8.3. Structures 55
8.4. Functions 56
8.4.1. audAddEntry() 56
8.4.2. audAddLowLevel() 57
8.4.3. audAddfChannelStr () 57
8.4.4. audfclose() 58
8.4.5. audfCvtEntryToOld() 58
8.4.6. audfCvtRawToEntry () 58
8.4.7. audfFindEntryDOS() 59
8.4.8. audfFindEntryStr() 60
8.4.9. audfFormatEntry () 61
8.4.10. audfNumRecs () 62
8.4.11. audfOpen() 62
8.4.12. audfReadEntry() 62
8.4.13. audfReadLowLevel() 63
8.4.14. audfReadOldStyle() 64
8.4.15. audfRecover() 64
8.4.16. audfVAddEntry() 65
9. Date and Time API 66
9.1. Overview 66
9.1.1. Year 2000 Compliance 66
9.1.2. Discrete Components Date and Time Format 66
9.1.3. Sort Date and Time String Format 67
9.1.4. Miscellaneous 67
9.2. Defines 68
9.2.1. DATSIZ 68
9.2.2. isLeapYear 68
9.2.3. LDATSIZ 68
9.2.4. PRND_*, PRNT_* 68
9.3. Variables 68
9.3.1. strMonths 68
9.3.2. strDays 69
9.4. Functions 69
9.4.1. addDaysToDate 69
9.4.2. dateDecode 69
9.4.3. dayFromDate 70
9.4.4. dcdate 70
9.4.5. difDate 71
9.4.6. difYear 71

9.4.7. lastDayOfMonth 71
9.4.8. ncdatel 72
9.4.9. ncedatl 72
9.4.10. prnDate 72
9.4.11. prnTime 74
9.4.12. sDateDecode 75
9.4.13. sDateDecodeDOS 75
9.4.14. sDateEncode 76
9.4.15. sDateEncodeDOS 76
9.4.16. sDecodeDT 77
9.4.17. sDecodeDTDOS 77
9.4.18. sTimeDecode 78
9.4.19. sTimeDecodeDOS 78
9.4.20. sTimeEncode 78
9.4.21. sTimeEncodeDOS 79
9.4.22. timeDecode 79
9.4.23. v2sdatl 80
9.4.24. validDate 80
9.4.25. validDateDOS 80
9.4.26. validTime 81
9.4.27. validTimeDOS 81

1. Overview

Worldgroup 3.12 incorporates many changes to the development environment. Several new environment variables are used to make compilation of your projects easier. As the development of Worldgroup 3.12 progressed, it became evident that the existing Worldgroup development environment was lacking in several places, most notably, every project being in one directory. What follows is everything that has changed since the last release of the Worldgroup development environment.

2. Development Tools

Galacticomm only supports Borland C++ 5.00 for re-compiling source code for both DOS and NT/95 servers.

Galacticomm only supports Microsoft Visual C++ 4.2 for re-compiling offline programs for NT/95.

Galacticomm only supports Microsoft Visual Basic 3.00 for re-compiling client side programs.

3. Development Environment Changes

With the advent of WebCast ProServer and ActiBase Server, it was clear our development environment needed an overhaul. Most notably, we needed to be able to #ifdef more than just MSG, DEF, and BCR files. Also, we could not be limited to #ifdef GCWINNT or #ifdef GCDOS. This realization resulted in the re-organization of the development environment and separating of each individual project into a directory of its own.

3.1. File Naming Conventions

Our DOS projects still adhere to the 8.3 naming conventions. However, you may find that some of our NT/95 only projects contain files that do not adhere to the 8.3 conventions.

3.2. Worldgroup Development Environment Requirements

This version of the Worldgroup Developers Kit requires a Windows 32 bit environment. Every developer utility is compiled for WIN32 and they all support long filenames.

3.3. Environment Variables

BC50	Root directory of the Borland C++ 5.0 directory (C:\BC5)
DOSEXT	Root directory of the DOS Extender for the DOS Environment (C:\RUN286)
WGDEV	Root directory of the Worldgroup development environment (C:\WGDEV)
WGDOS	Root directory of the Worldgroup DOS runtime (C:\WGSERV)
WGNT	Root directory of the Worldgroup NT/95 runtime (C:\WGSERVNT)
BUDOS	Used to create VIR files for the DOS environment (Explained later in this document)
MVC42	Root directory of Microsoft Visual C++ 4.2 projects (C:\MSDEV)
VB3	Root directory of Visual Basic 3.0 (C:\VB)

3.4. Development/Runtime Tree

Directory	Description/What is contained in them
- WGDEV	Main development directory
- INC	Shared header files
- BIN	Development executables and batch files (formerly WGTOOLS)
- LIB	DEF files and all shared LIB files
- HELP	Developer documentation
- SRC	Source code broken down into single projects
- VCPROJ	Microsoft Visual C++ projects

3.5. The \WGDEV\SRC directory

The most notable change to the development environment is in the source directories. Formerly \WGDEV\WGSRC, this directory was renamed and separated into multiple projects. The new directory structure under \WGDEV\SRC looks like this:

Directory	Description/What is contained in them
- \WGDEV\SRC	Source directory
- API	(gcommlib, galme, galsmb...)
- APPS	(galfil, galpnq, galrsy, galreg...)
- DEBUG	(galdebug, galdebgcg...)
- ICSRC	(galtnt, galtcpip, galftp...)
- SERVER	
- WGSERVER	(wgserver, wgsignup, galsupah...)
- UTILS	(wgsdraw, wgsmtree, wgsfndo...)
- OFFLINE	(wgscmaps, wgscolor, wgsmref...)
- REBUILD	(wgsrbk, wgsrbt...)
- DEVUTILS	(galcopy, galport, galprod...)

3.6. PATH Statement

The \WGDEV\WGTOOLS directory no longer exists. Instead, your path statement should be changed to include \WGDEV\BIN.

3.7. Files contained in each project

Each project directory contains ALL files pertaining to that project (this includes all source: .C and .H, .MSG, .DEF, .BCR, .HTML, .GIF, and .JPG). As part of the compilation (mknt, mkdosp...), each .MAK file compiles/copies files to their proper build directories, most of this being automated by GCRULES.MAK. Files that are shared in multiple projects appear in both projects. Common .H files appear in their project as well as \WGDEV\INC. Common .DEF files appear in their project and \WGDEV\LIB. This may seem like a large undertaking, however, using version control to track source files simplifies this process.

3.7.1. #ifdef'ing files

Worldgroup 3.00 allowed you to #ifdef .MSG, .DEF, and .BCR files, and the utility WGSSYNC.EXE parsed these files during compilation. However, you were only allowed to use GCWINNT and GCDOS for defines. With Worldgroup 3.12, you can now #ifdef .MDF, .MSG, .HTM, .CVT, and .DMD. Additional files may be added on a per-project basis. You are no longer limited to using GCWINNT or GCDOS. Any #ifdef you use in source code may also be used in the parse files. For a description of the limitations of #ifdef'ing files, please see the section on GALPARSE.EXE under New Development Utilities.

3.7.2. Making Worldgroup 3.00 .MAK files work for 3.12

Minor changes are required in order for Worldgroup 3.00 .MAK files to work under the new environment. The following is a step by step list of what needs to change:

1. The line *!include $(WGDEV)\wgsrc\GCRULES.MAK* needs to be replaced with *!include $(WGDEV)\src\GCRULES.MAK*.

2. Any references to SRCDIR and SRCDIROS should be removed. All files for projects are contained in the project directory. There is no need for these macros anymore.

3. The $(MODEL) macro for DOS compilations now gets set to "LARGE" or "PHARLAP" instead of "L" or "P".

4. Any references to the $(CC) macro should be changed to $(PARSEOPTS). The $(PARSEOPTS) macro is passed on to the GALPARSE.EXE which is described later in this document.

5. All references to $(DLIBDIROS) should be removed. GCRULES.MAK defines where all .LIB files are located.

6. All CD commands should be removed from the make procedure. There should be no reason to change directories. GCRULES.MAK has all paths set up correctly.

7. If your project contains a .DEF file, the line that has *DEFILE = (file)* needs to be changed *to DEFILE = $(TEMPDIR)\(file)*. Your .DEF file in your project can now contain #ifdef's. As part of the $(GCBUILD) macro, your .DEF files are parsed to $(TEMPDIR).

8. There is a new macro that needs to be added to the beginning of the dependencies list. $(GCBUILD) forces all pre-build parsing to take place. The $(GCBUILD) macro can be found inside of GCRULES.MAK in the \wgdev\src directory and is discussed later in this document.

9. The $(GCBUILD) is a dependency of your project that always changes. Because of this, every time you re-compile, the make utility thinks something has changed, and forces a re-link. To solve this problem, do the following:

Instead of having:

```
galfil.dll:                     \
        $(GCBUILD)              \
        galfil.h                \
        $(OBJFILES)    ...
```

Do the following:

```
start:                          \
        $(GCBUILD)              \
        galfil.dll

galfil.dll:                     \
        galfil.h                \
        $(OBJFILES)     ...
```

The following is the Worldgroup 3.00 .MAK file for Cross-Wordz, followed by the changes required for this .MAK file to work properly in the 3.12 environment:

```
#####################################################################
#  This makefile generates TTICWD.DLL                               #
#####################################################################
!include $(WGDEV)\wgsrc\GCRULES.MAK

SRCDIR = $(WGDEV)\cwdsrc
SRCDIROS = $(SRCDIR)\$(OS)

!if $(OS) == DOS
!       if $(MODEL) == L
!               error TTICWD not supported in Large Module Under DOS!
!       endif
!endif

DLLLIB =                                \
        $(DLLLIB)                       \
        $(DLIBDIROS)\wgserver.lib       \
        $(DLIBDIROS)\galgsbl.lib

OBJFILES =                              \
        tticwd.OBJ                      \
        cscwd.OBJ                       \
        cwdboth.OBJ

tticwd.dll:                                                     \
        tticwd.h                                                \
        $(OBJFILES) $(DLLLIB) $(DLIBDIROS)\$(DEFILE)
        CD $(OBJDIR)
        $(LINK) $(LIBCMD) @&&|
$(PRODLL) $(OBJFILES)
$(RUNDIR)\$&
$(RUNDIR)\$&
$(DLLLIB) $(LINKDLL)
$(DLIBDIROS)\$(DEFILE)
|
        CD $(RUNDIR)
```

This is what the Worldgroup 3.12 version of the .MAK file looks like, with the changes highlighted:

```
#####################################################################
#  This makefile generates TTICWD.DLL                               #
#####################################################################
!include $(WGDEV)\src\GCRULES.MAK

!if $(OS) == DOS
!       if $(MODEL) == LARGE
!               error TTICWD not supported in Large Module Under DOS!
!       endif
!endif

DLLLIB =                \
        $(DLLLIB)       \
        wgserver.lib    \
        galgsbl.lib

OBJFILES =              \
        tticwd.OBJ      \
        cscwd.OBJ       \
        cwdboth.OBJ

start:                  \
        $(GCBUILD)      \
        tticwd.dll

tticwd.dll:             \
        tticwd.h        \
        $(OBJFILES)     \
        $(DLLLIB)       \
        $(DEFILE)
        $(LINK) $(LIBCMD) @&&|
$(PRODLL) $(OBJFILES)
$(RUNDIR)\$&
$(RUNDIR)\$&
$(DLLLIB) $(LINKDLL)
$(DEFILE)
|
```

3.7.3. New .MAK file macros

The following is a list of new macros you may use in your .MAK files to make building projects easier:

CVTFILDIR - The directory to parse .CVT files to. Defaults to $(RUNDIR)\wgsdfcvt

HTMLDIR - Directory to parse .HTM files to. Defaults to $(RUNDIR)\webpages, however, for an ActiveH project such as File Libraries, you may change this to $(RUNDIR)\galacth\galfilah.

GIFDIR - Directory to copy all .GIF and .JPG files to. Defaults to $(RUNDIR)\webages\images.

3.7.4. The $(GCBUILD) macro and GCBUILD directories

This macro expands to the filename tmpbld.bld. This file is a dependency in every project that forces certain events to occur for every compilation. Its main function is to track the list of $(PARSEOPTS) used for the last compilation and force a rebuild of the entire project if the list differs from compilation to compilation. Having code compile in many different ways through the use of #ifdef's is great, however, if you compile something one way, then turn around and compile with a different set of #ifdef's, your project

does not get re-built, only re-linked. To avoid cross-compiling sources in this manner, use the $(GCBUILD) macro.

Instead of separating your .OBJ files into separate directories off the root \WGDEV directory, each project has a directory beneath it called GCBUILD. The GCBUILD directory has the following directories beneath it:

WNT	- Windows NT compilation files.
WNTD	- Windows NT debug compilation files.
DOSP	- DOS Pharlap compilation files.
DOSPD	- DOS Pharlap debug compilation files.
DOSL	- DOS large compilation files.
DOSLD	- DOS large debug compilation files.

These directories contain all relevant compile time files (.OBJ, .DEF, .LIB...). Adding the $(GCBUILD) macro to the beginning of your project forces the following:

1. Checks the list of previous $(PARSEOPTS) saved in the GCBUILD directories. If they are identical, nothing else happens. This is to insure that all relevant files are re-parsed with the proper $(PARSEOPTS) macro.

If the GCBUILD directory does not exist, the following happens:
2. All files are parsed to the GCBUILD directory.

3. .GIF, .JPG, and .BIN files are copied to the appropriate directories.

The GCBUILD directories may be deleted at any time. They only contain files that are needed for compile time. Deleting the directory does not harm your project.

Projects will not compile properly if the $(GCBUILD) macro is left out. Inserting it insures that the projects are compiled the way you want them to be.

3.7.5. Creating .VIR files from within .MAK files.

You now have the ability to create .VIR files as part of your make procedures. Adding another dependency to your .MAK file forces .VIR files to be built on the fly. The following lines be added to accomplish this:

```
VIRFILES =              \
        file1.vir       \
        file2.vir       \
        file3.vir
```

then adding $(VIRFILES) as one of your DLL dependencies forces the .VIR files to be built. You must have an equivalent .BCR file located in your project directory. The creation of .VIR files under the Windows NT/95 server environment is set up for you by default. However, if you wish to create .VIR files for DOS, you must own a copy of the utility BUTIL.EXE included with the Btrieve Developers Kit. If you own a copy of BUTIL.EXE, the environment variable BUDOS should be pointed to it. If BUDOS is in your environment .VIR files are created for DOS. Example:

```
set BUDOS=C:\BUTIL\BUTIL.EXE
```

3.8. Compilation Batch Files

Most compilation batch files have a matching debug batch file. The difference between the two is the debug version passes -DDEBUG to MAKE. Both *mkntd galfil* and *mknt galfil -DDEBUG* do the exact same thing. The following is a list of batch files used for the compilation of Worldgroup source:

3.8.1. MKNT.BAT and MKNTD.BAT

Compiles Borland C files for Worldgroup NT/95. The environment variables WGNT, WGDEV, and BC5 are required for these batch files.

3.8.2. MKDOSL.BAT and MKDOSLD.BAT

Compiles Borland C files for Worldgroup DOS in LARGE mode. The environment variables WGDOS, WGDEV, and BC5 are required for these batch files.

3.8.3. MKDOSP.BAT and MKDOSPD.BAT

Compiles Borland C files for Worldgroup DOS in PHARLAP mode. The environment variables WGDOS, WGDEV, BC5, and DOSEXT are required for these batch files.

3.8.4. MK31V.BAT and MK31VD.BAT

Compiles 16 bit Borland C files for Windows. When these batch files are used, VB-Runtime errors replace the normal Galacticomm catastro functions. The environment variables WGMAN, WGDEV, and BC31 are required for these batch files.

3.8.5. MK31W.BAT and MK31VW.BAT

Compiles 16 bit Borland C files for Windows. When these batch files are used, catastros simply cause the windows DLL to terminate. The environment variables WGMAN, WGDEV, and BC31 are required for these batch files.

3.8.6. MKMVC.BAT

Compiles Microsoft Visual C++ 4.2 files from the command line using a wrapping .MAK file. Examples of this can be found in the \WGDEV\VCPROJ project directories. The environment variables WGNT, WGDEV, and MVC42 are required for this batch file.

3.8.7. MKVB.BAT and MKVBC.BAT

Compiles Microsoft Visual Basic 3.0 files from the command line using a wrapping .MAK file. Examples of this can be found in the \WGMAN project directories. Both batch files require the WGDEV and VB3 environment variables. The MKVBC.BAT file requires the WGDOS and WGNT environment variables so that it can copy the client executables to their proper directories. For client builds, MKVBC.BAT requires that the .MDF file of the client you are building be located in your project directory.

3.8.8. MAKEWGS.BAT

The new usage for MAKEWGS.BAT is as follows:

MAKEWGS (What) (Type)

What	Description
ALL	Builds everything required for server.
SERVER	Builds only server code.
SRVUTIL	Builds only server utilities.
OFFLINE	Builds all offline utilities.
DEVUTIL	Builds all developer utilities.
API	Builds all API relate libraries.
APP	Builds all Worldgroup Apps.
GALME	Builds all messaging engine files.
CLIENT	Builds all client files.
MVC	Builds Microsoft Visual C++ apps.
JAVA	Builds Java Apps.

Type	Description
DOS	Build Worldgroup for DOS.
DOSD	Build Worldgroup for DOS, debug mode.
WNT	Build Worldgroup for NT.
WNTD	Build Worldgroup for NT, debug mode.

You can pass defines to MAKEWGS.BAT. Here is an example of compiling all Windows NT source with source debugging:

makewgs all wntd -DSRCDBG

3.9. New Development Utilites

For complete functionality of each new executable simply run the programs. Below is a list of all new programs added to the development environment.

3.9.1. GALCOPY.EXE

Generic utility used to copy files. Mainly created because of differences between the Windows 95 and Windows NT copy commands. It is used by GCRULES.MAK to copy files to proper directories.

3.9.2. GALMERGE.EXE

Message file merging utility. Used by GCRULES.MAK to merge your project .MSG files with existing .MSG files.

3.9.3. GALMKDIR.EXE

Generic mkdir program. It can take a nested directory path and create it without error. Used by GCRULES.MAK to make sure all directories needed are created before compile time.

3.9.4. GALPARSE.EXE

Utility used to parse all (.MSG, .MDF, .HTM...) files. This utility is used in conjunction with the $(PARSEOPTS) macro in .MAK files. All options that are passed on to the compiler in your .MAK files are passed on by adding to the $(PARSEOPTS) macro. This macro is used in conjunction with other internal defines to parse all needed files.

3.9.5. GALSYNC.EXE

Used by the $(GCBUILD) macro in your .MAK files to make sure that your project is being compiled with the same #defines. A complete description of how this works can be found in the section on the $(GCBUILD) macro.

3.9.6. GALPROD.EXE

Timestamping utility similar to touch.exe. However, this utility allows you to touch a file with a new date as well as time.

4. Data File Conversion

The previous data file conversion API's and .CVT file structures had many shortcomings. Most notably, it became harder to create .CVT files that could properly track and upgrade multiple versions of one data file between both DOS and NT/95. Certain situations would cause data files to be converted more than once, thereby destroying the data contained in the file. These and many other issues forced us to create a new data file conversion API.

4.1. Backward compatibility

The new data file conversion API does not remove old functions. Existing .CVT files and custom conversions will continued to be supported through the new API, with a few exceptions:

1. .CVT files that are used to convert a file to Worldgroup NT/95 and specify "DestPacked: YES" are no longer supported. Because of shortcomings in the old system, there is no way to say with 100% certainty that a .CVT file is intended to convert to a Worldgroup NT/95 file. You have to change all .CVT files that fall into this category to the new style.

2. Custom conversion programs will only be passed one of 2 strings as the conversion type: "2.0 to NT", "3.0 to NT". No modifications are being made to pass "2.0 to 3.1 NT" or any equivalent. If you need to do data file conversions that may require this information, you need to change to the new style custom conversions.

4.2. WGSDFCVT.EXE

This utility was totally re-written. The old interface that asked you which files you wanted to convert no longer exists. Instead, the new interface scans all .CVT files and decides if a conversion needs to take place. If a data conversion is required, the operator is asked to specify the directory he would like to back up files to. Leaving this option blank forces no backup. No other user intervention exists, and a screen of information regarding the files wgsdfcvt is converting appears. If no datafile conversion needs to take place, nothing is displayed to the user.

4.3. Conversion file theory

The old conversion file API's worked on the premise that if the "From" file existed and the .CVT file was in the directory, then it needed conversion. Because of this, installation programs needed to have some smarts built into them, to decide which copies of the .CVT files should be installed.

The new API uses one .CVT file per data file. Each .CVT file contains either a custom conversion program, or the entire history of the data file, including the current and older versions of the file layouts, thereby eliminating the need to have an intelligent installation program. WGSDFCVT can now focus on converting all older versions of a file to its newest form.

Either through .CVT file, or through custom conversion, a list of all versions of a data file are submitted to the conversion API. The conversion API goes through this list and determines the number of previous versions that exist. Under normal circumstances there is only one previous file. However, if multiple files exist, it converts the one with the most recent date and time. As the need arises to make changes to a data file, additions to the existing .CVT file or custom conversion program simplify the conversion process.

As an example, a .CVT file might provide information on the history of WGSUSR2.DAT in such a way that WGSDFCVT.EXE creates a list of files much like this:

DOS Platform	NT/95 Platform
BBSUSR.DAT (pre-Worldgroup 3.00) WGSUSR2.DAT (Current Version)	WGSUSR2.DAT (Current Version)

Installing and converting on NT/95:
WGSDFCVT.EXE searches for existing copies of BBSUSR.DAT and WGSUSR2.DAT. If BBSUSR.DAT exists, it reads a record out of the database, converts it in memory to the WGSUSR2.DAT DOS form, then in memory to the WGSUSR2.DAT WNT form and saves it in the new datafile.

Installing and converting on DOS:
WGSDFCVT.EXE searches for an existing copy of BBSUSR.DAT. If BBSUSR.DAT exists, it reads a record out of the database, converts it in memory to the WGSUSR2.DAT DOS form, then saves it in the new datafile.

.CVT files are used mainly to convert data files from one platform to another. If your conversions require more than simple re-aligning of fields (adding fields for instance), then you can create a custom conversion program and supply your own set of conversion functions.

4.4. .CVT file formats

The .CVT file formats are completely different from the older version. There is no longer a need to have multiple .CVT files for one data file. Each .CVT file contains all information pertaining to the life of its associated data file. Numerous examples of the new .CVT file structure can be found in \WGSERV\WGSDFCVT. The new .CVT file format is as follows:

```
[;]
Description: <description>

Developer: <developer name>

[CustomConversion: <conversion file>]

[
BeginFile: <platform>,<file>[,<platform>,<file>...]
<tagname 1>       TYPE=<field type> [ELEMS=<length>]  [VARIABLE]
...
<fldtagN> TYPE
EndFile

[
BeginFile: <platform>,<file>[,<platform>,<file>,...]
...
]
]
```

4.4.1. The ; operator

The semi-colon ";" is used as a remark. All text after the semi-colon is ignored.

4.4.2. Description:

This is a required field. Include a description of the data file that this .CVT file is associated with. Text is truncated past 30 characters of the description.

4.4.3. Developer:

This is a required field. Include the name of your company. This field is used to determine the difference between the new and old .CVT files. Text is truncated past 30 characters of the developer name.

4.4.4. CustomConversion:

Use this field to specify a custom conversion program for a data file. No other fields should be included if a custom conversion is present. Custom conversion programs are described later in this document.

4.4.5. BeginFile: and EndFile

These are required fields for any NON-Custom Conversion .CVT file. For every BeginFile:, there is exactly one EndFile. Data Fields are required between each occurrence of BeginFile: and EndFile, and are discussed later in this document. The BeginFile: Line contains a list of platforms and data file names separated by commas. Only one platform/data file is required, however, you may include as many as you want. Each BeginFile:/EndFile combination is a list of data fields contained within a data file. It is quite possible that a data file has the same set of fields for both DOS and WNT, the only difference being the data alignment (DOS being packed, and WNT being unpacked). The following is an example taken from WGSUSR.CVT:

BeginFile: DOS, WGSUSR2, WNT, WGSUSR2
(data fields here)
EndFile

This combination inside a .CVT file specifies that the data fields contained within BeginFile: and EndFile belong to the files: WGSUSR2.DAT on the DOS platform, and WGSUSR2.DAT on the WNT platform.

The only valid platform identifiers are: DOS for DOS Btrieve 5.x datafiles, and WNT for Btrieve 6.x data files. The data file conversion API assumes the following: DOS platforms are packed/little endian, and WNT platforms are unpacked/little endian. If your conversion requires that these defaults not be followed, you must create a custom conversion and bypass these defaults.

4.4.6. Data Fields

Data Fields are similar to their old versions. You must have at least one data field for every BeginFile:/EndFile combination. Data Fields are defined as follows:

<Tag Name> TYPE=<Field Type> [ELEMS=<length>] [VARIABLE]

Tag Name
A 19 character or less name for the data type. This generally is the same name as the variable it represents in the data structure. There can only be one occurrence of each Tag Name within each BeginFile:/EndFile combination.

TYPE
Valid field types are: CHAR, SHORT, LONG, FLOAT, DOUBLE, OPAQUE, and STRUCT.

ELEMS
This specifies the number of occurrences of this field type in this data field. For instance, if you are creating a field type for the following variable:

SHORT MyString[500];

your data field would look like this:

```
MYSTRING            TYPE=SHORT          ELEMS=500
```

ELEMS is not a required field and omitting it sets the number of elements to 1.

VARIABLE
The keyword VARIABLE is used to designate a variable length field. Variable length fields only work on the last Data Field in your list. You must include the ELEMS field when you use the VARIABLE keyword, however, the ELEMS field should specify the maximum number elements that could be present in this variable field.

4.4.7. TYPE=STRUCT and EndStruct

The previous version of the data file API had two serious flaws: You could only nest structures one deep, and you could not remove data fields that were within structures. This is no longer the case. Structures are now defined in the same way that BeginFile:/EndFile are used. The following is WGSMENU.CVT. It has an example of variable length structures:

```
BeginFile: DOS,BBSMENU
MENUTYPE     TYPE=CHAR
PAGENAME            TYPE=CHAR           ELEMS=16
PARENTPAGE          TYPE=CHAR           ELEMS=16
FILENAME            TYPE=CHAR           ELEMS=80
MODULENAME          TYPE=CHAR           ELEMS=25
COMMANDSTG          TYPE=CHAR           ELEMS=60
GOLOCK              TYPE=CHAR           ELEMS=16
FLAGS               TYPE=SHORT
MENUTITLE           TYPE=CHAR           ELEMS=43
NPAGES              TYPE=SHORT
ICHANGE             TYPE=LONG
SPARE               TYPE=CHAR           ELEMS=70
PGLINK        TYPE=STRUCT         ELEMS=25            VARIABLE
        POSITION            TYPE=SHORT
        SELCHR              TYPE=CHAR
        ICONAME             TYPE=CHAR           ELEMS=9
        DESTPAGE            TYPE=CHAR           ELEMS=16
        OPTDSP              TYPE=CHAR
        KEYREQ              TYPE=CHAR           ELEMS=16
        SHORTD              TYPE=CHAR           ELEMS=31
        LONGD               TYPE=CHAR           ELEMS=50
EndStruct
EndFile
```

All field names are unique. Field names within substructs must have different names than any other fields in your .CVT files. You can not have a field named “KEY” inside a structure and outside a structure.

4.5. New Data File Conversion API

All previous data file conversion API’s still exist and are supported, however, you should migrate your software to the new API functions. The new API’s and structures are located in the file DFCAPI2.H.

4.5.1. DFCLIST structure

```
typedef struct tagDFCLIST {        /*  Platform Specific list of files     */
     INT Platform;                 /*  Operating system for file list      */
     pDFCFILE file;                /*  Pointer to the file information     */
```

```
} DFCLIST,*pDFCLIST;
```

Used to create a list of files that the dfc library should check for conversion. An array of these structures are passed to dfcRegister(). Your DFCLIST array must end with a platform of DFC_END and a file of NULL.

Platform
Valid values are DFC_DOS, DFC_WNT, and DFC_END. DFC_END is used to mark the end of the list.

file
pointer to an array of DFCFILE structures that contain a list of files specific to the given platform. This can be NULL when the Platform is DFC_END.

4.5.2. DFCFILE structure

```
typedef struct tagDFCFILE {        /* Datafile information                */
     CHAR Name[GCMAXPTH];          /*  Name of the datafile               */
     pDFCFUNCTION cnvfunc;         /*  Conversion function                */
     pDFCDATAFIELDS cnvfields;     /*  Conversion fields                  */
} DFCFILE,*pDFCFILE;
```

Used to create an array of files that apply to a specific platform. The DFCFILE structure is directly related to the DFCLIST structure. All DFCFILE structure arrays must end with a blank name.

Name
The name of the data file without the extension. A blank name specifies the end of a file list.

cnvfunc
Pointer to a conversion function or NULL to use dfcConvertDefault. This function is used to convert data from the previous version of the file. If this pointer is for the first file in a list it is ignored.

cnvfields
Pointer to an array of DFCDATAFIELDS structures. This array contains information on each data field within the data file similar to those used in .CVT files.

4.5.3. DFCDATAFIELDS structure

```
typedef struct tagDFCDATAFIELDS {  /* Datafile fields definition          */
     CHAR Name[DFC_NAMSIZ];        /*  Name of the field                  */
     UINT Type;                    /*  Type of the field                  */
     INT nelems;                   /*  Number of elements                 */
     struct tagDFCDATAFIELDS *substruct;/*  Pointer to sub structure      */
} DFCDATAFIELDS,*pDFCDATAFIELDS;
```

Used to create a set of data fields contained within a data file. This structure is used in conjunction with the DFCFILE structure. All DFCDATAFIELDS must end with a blank name and a type of DFCFLD_END.

Name
A 19 character or less name for the data field. Leave this field blank to specify the end of the list.

Type
Valid types are: DFCFLD_CHAR, DFCFLD_SHORT, DFCFLD_LONG, DFCFLD_FLOAT, DFCFLD_DOUBLE, DFCFLD_OPAQUE, DFCFLD_STRUCT.

For variable length fields use: DFCFLD_VCHAR, DFCFLD_VSHORT, DFCFLD_VLONG, DFCFLD_VFLOAT, DFCFLD_VDOUBLE, DFCFLD_VOPAQUE, DFCFLD_VSTRUCT.

DFCFLD_END is used to specify the end of the list.

nelems
The number of elements in this field. For variable length fields, specify the maximum number of fields.

substruct
If the field type is DFCFLD_STRUCT or DFCFLD_VSTRUCT, you must supply another array of DFCDATAFIELDS for that structure, otherwise this parameter is NULL.

4.5.4. DFCOFFSETS structure

```
typedef struct tagDFCOFFSETS {      /* Offset information structure         */
     CHAR Name[DFC_NAMSIZ];         /*  Name of the variable                */
     UINT Type;                     /*  Type of record it is                */
     INT nelems;                    /*  Number of elements                  */
     UINT Offset;                   /*  Offset within data record           */
} DFCOFFSETS,*pDFCOFFSETS;
```

A bunch of internal structures are created when you call dfcRegister() with your file list. The DFCOFFSETS structure is an internal list past to you as part of the DFCFILEINFO structure. This contains information on each one of your data fields. The DFCOFFSETS structure and the DFCDATAFIELDS structures are closely related. This structure is never created by you. You can obtain pointers to this structure from within your conversion functions by calling dfcFindOffset().

Name
The name of the data field. Equivalent to the names from your DFCDATAFIELDS structure.

Type
The type of data field. Equivalent to the types from your DFCDATAFIELDS structure.

nelems
The number of elements in the data field. Equivalent to the number of elements from your DFCDATAFIELDS structure.

Offset
The offset within the data that this field is at. If you have the following structure:

CHAR String[20];
CHAR Test[10];

and you called dfcFieldOffst() for the TEST data field, the offset would be 20. The offset does account for alignment. DOS versions and WNT versions of the same data fields may produce different offsets.

4.5.5. DFCFILEINFO structure

```
typedef struct tagDFCFILEINFO {     /*  File info to pass to conversions    */
     INT Platform;                  /*  Current system Platform             */
     CHAR Name[GCMAXPTH];           /*  Conversion file                     */
     INT Type;                      /*  Conversion macros (CVTBIGEND...)    */
     pDFCDATAFIELDS Fields;         /*  Pointer to the fields in data       */
     pDFCOFFSETS Offsets;           /*  Pointers to offset information      */
     UINT Offsetsn;                 /*  Number of offsets                   */
     struct flddef *fdef;           /*  Field definition array (set up)     */
     CHAR *Data;                    /*  Pointer to actual data              */
     size_t DataLength;             /*  Length of the data                  */
     struct flddef** fdefs;         /*  Used to allocate memory for fdef    */
     UINT fdefsn;                   /*  Number of field defs                */
} DFCFILEINFO,*pDFCFILEINFO;
```

If you have created conversion functions, you are passed two pointers to this structure. The structure contains a wealth of information about specific files being converted.

Platform
The platform this DFCFILEINFO structure belongs to, either DFC_DOS or DFC_WNT.

Name
The name of the file this DFCFILEINFO structure is associated with.

Type
These are the data types for Platform (i.e. Big/Little Endien, Packed/Unpacked). This variable can be used in calls to the cvtData() functions.

Fields
A pointer to the array of DFCDATAFIELDS structures you passed to dfcRegister().

Offsets
A pointer to an array of DFCOFFSETS structures created based on your DFCDATAFIELDS structures. These are most easily accessed through the use of the function dfcFindOffset().

Offsetsn
The number of offsets contained in the *Offsets* array.

fdef
A pointer to a field definition array created based on the array of DFCDATAFIELDS.

Data
If you are passed this information in the *from* parameter of a conversion function, it contains the data read directly from the data file. If you are passed this information in the *to* parameter of a conversion function, it contains a buffer that is allocated to the largest amount of bytes a record can take. You can use that buffer to store your converted data.

DataLength
This serves two purposes. In the *from* parameter of a conversion function, it is the size in bytes of the data present in the Data field. In the *to* paremater of a conversion function, it is the size in bytes of the supplied Data field. In variable length records, the DataLength field from the to parameter should be changed to reflect the correct size of the record being inserted.

fdefs
Used to produce the fdef field. This field should not be used or modified.

fdefsn
The number of fdefs in the array. This field should not be used or modified.

4.5.6. DFCPFCONVERT structure

```
typedef struct tagDFCPFCONVERT {    /*  Platform conversion                */
     INT PlatformFrom;              /*  Platform converting from           */
     INT PlatformTo;                /*  Platform converting to             */
     pDFCFUNCTION cnvfunc;          /*  Pointer to conversion function     */
} DFCPFCONVERT,*pDFCPFCONVERT;
```

Used to specify a list of conversion functions that convert data from one platform to another. Currently the only combination of platforms supported is DOS to WNT.

PlatformFrom
Valid values are DFC_DOS, DFC_WNT, and DFC_END. Currently only DFC_DOS is supported as part of the PlatformFrom field. DFC_END is used to specify the end of a list.

PlatformTo
Valid values are DFC_DOS, DFC_WNT, and DFC_END. Currently only DFC_WNT is supported as part of the PlatformTo field.

cnvfunc
Pointer to a conversion function used to convert data from PlatformFrom to PlatformTo.

4.5.7. dfcAskBackup()

```
VOID
dfcAskBackup(VOID);                 /*  Ask the backup directory           */
```

Ask the user what directory they wish to back files up to. If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

4.5.8. dfcBackupDirectory()

```
CHAR *
dfcBackupDirectory(VOID);           /*  Returns pointer to backup directory */
```

If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

Returns
Pointer to the directory the user has asked to back files up.

4.5.9. dfcBackupRequired()

```
GBOOL
dfcBackupRequired(VOID);            /*  Is a backup required?              */
```

Returns
TRUE if the user has requested to back up their data files,
FALSE if they have not.

4.5.10. dfcConvertDefault()

```
DFCFUNCTION                         /*  Pointer to new data                */
dfcConvertDefault(                  /* Default conversion function         */
pDFCFILEINFO from,                  /*  File information converting from   */
pDFCFILEINFO to);                   /*  File information converting to     */
```

Default conversion function. This function is called if you do not supply your own custom conversion function.

from
A pointer to a DFCFILEINFO structure that gives you information on the file you are converting from. See the section titled "DFCFILEINFO" for more details.

to
A pointer to a DFCFILEINFO structure that gives you information on the file you are converting to. See the section titled "DFCFILEINFO" for more details.

Returns
A pointer to the converted data.

4.5.11. dfcConvertPFDefault()

```
DFCFUNCTION                         /*  Pointer to new data                */
dfcConvertPFDefault(                /* Default Platform conversion         */
pDFCFILEINFO from,                  /*  File information converting from   */
pDFCFILEINFO to);                   /*  File information converting to     */
```

Platform default conversion function. This function is called only when converting from one platform to another (i.e. DOS to WNT), unless you have provided your own platform conversion function.

from
A pointer to a DFCFILEINFO structure that gives you information on the file you are converting from. See the section titled "DFCFILEINFO" for more details.

to
A pointer to a DFCFILEINFO structure that gives you information on the file you are converting to. See the section titled "DFCFILEINFO" for more details.

Returns
A pointer to the converted data.

4.5.12. dfcDisplayBackup()

```
VOID
dfcDisplayBackup(VOID);             /*  Display backup notice              */
```

Displays a notice that says "Backing up data files". If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

4.5.13. dfcDisplayInfo()

```
VOID
dfcDisplayInfo(                     /* Display DFC information             */
const CHAR *Description,            /*  Description to display            */
const CHAR *FileFrom,               /*  File converting from              */
const CHAR *FileTo,                 /*  File converting to                */
ULONG Records);                     /*  Number of records to display      */
```

Updates display information about your custom conversion on the screen. If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

Description
A 30 character description of the data file being converted.

FileFrom

Name of the data file that is being converted.

FileTo
Name of the data file you are converting to.

Records
Total number of records that are in the data file.

4.5.14. dfcDisplayStats()

```
VOID
dfcDisplayStats(                    /*  Display the stats bar              */
ULONG TotalRecords,                 /*  Total records in conversion        */
ULONG DoneRecords);                 /*  Number of records completed        */
```

Used to update the number of records converted. If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

TotalRecords
Total number of records in the database.

DoneRecords
Total number already converted.

4.5.15. dfcDisplayShutdown()

```
VOID
dfcDisplayShutdown(VOID);           /*  Release captured video screen      */
```

Closes all windows that may have been opened while using dfcDisplay... functions. If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

4.5.16. dfcError()

```
VOID
dfcError(                           /*  Print an error to screen           */
const CHAR *stg,                    /*  String to print                    */
...);                               /*  Extra arguments                    */
```

A generic error handling routine. Calling this function causes your custom conversion program to immediately end.

stg
A message to display upon the termination of your program.

4.5.17. dfcFieldSize()

```
UINT                                /*  Number of bytes of field size       */
dfcFieldSize(                       /* Gets field size of data type         */
UINT Type);                         /*  Type of datafield                   */
```

Gets the size in bytes of a given data field type.

Type
Valid types are: DFCFLD_CHAR, DFCFLD_SHORT, DFCFLD_LONG, DFCFLD_FLOAT, DFCFLD_DOUBLE, DFCFLD_OPAQUE, DFCFLD_STRUCT, DFCFLD_VCHAR, DFCFLD_VSHORT, DFCFLD_VLONG, DFCFLD_VFLOAT, DFCFLD_VDOUBLE, DFCFLD_VOPAQUE, DFCFLD_VSTRUCT

Returns
The size in bytes of the data type passed, or 0 if the data type is not found.

4.5.18. dfcFilePlatform()

```
INT                                 /*  DFC_DOS, DFC_WNT, DFC_END           */
dfcFilePlatform(                    /* Files platform                       */
const CHAR *file);                  /*  Pointer to file to check            */
```

Checks a data file and returns what platform the file belongs to (DOS/WNT). There are limitations to this function. If the program that is calling this function is compiled for DOS. It does not recognize WNT data files. We do not support data file conversion from Worldgroup NT/95 to Worldgroup DOS. If you are using dfcRegister to convert your data files, this function is automatically called by the internal API's.

File
Data file to check the platform on.

Returns
DFC_DOS for a btrieve 5.x data file.
DFC_WNT for a btrieve 6.x data file.
DFC_END on error.

4.5.19. dfcFindOffset()

```
pDFCOFFSETS                         /*   returns ptr to field offset info   */
dfcFindOffset(                      /* find offset of a particular field    */
pDFCFILEINFO info,                  /*   file information converting from   */
const CHAR *FieldName);             /*   name of field to find              */
```

Find the offset of a data field within a DFCFILEINFO structure. This function is primarily used within a conversion function to get information on a particular field.

info
Pointer to a DFCFILEINFO structure. Usually either the *to* or *from* fields in a conversion function.

FieldName
Name of the data field you want the offset structure for. These names are contained in the DFCLIST structure you passed to dfcRegister().

Returns
A pointer to a DFCOFFSETS structure, or NULL if FieldName does not exist.

4.5.20. dfcGetFDA()

```
struct flddef *                    /*  Pointer to FDA structure           */
dfcGetFDA(                         /* Get an FDA from type of convert     */
UINT Type);                        /*  Type of option                     */
```

Gets an FDA for a given data type.

Type
Valid types are: DFCFLD_CHAR, DFCFLD_SHORT, DFCFLD_LONG, DFCFLD_FLOAT, DFCFLD_DOUBLE, DFCFLD_OPAQUE, DFCFLD_STRUCT, DFCFLD_VCHAR, DFCFLD_VSHORT, DFCFLD_VLONG, DFCFLD_VFLOAT, DFCFLD_VDOUBLE, DFCFLD_VOPAQUE, DFCFLD_VSTRUCT

Returns
A pointer to a valid flddef structure or NULL. This structure could be used as part of a call to cvtData().

4.5.21. dfcRegister()

```
GBOOL                              /*  TRUE if conversion took place      */
dfcRegister(                       /*  Register and search for files      */
const CHAR *descrp,                /*  Description name                   */
const DFCLIST *dfc,                /*  pointer to all DFCFILES            */
const DFCPFCONVERT *pfconv,        /*  Platform Converter                 */
INT Key,                           /*  Key number to use (-1 for physical) */
INT Direction);                    /*  Direction (DFC_FORWARD/DFC_BACKWARD)*/
```

Notifies the data file conversion API of a list of files to check for conversion.

descrp
A 30 character or less description of the data file to check. Anything past 30 characters is truncated.

dfc
A pointer to a DFCLIST structure that contains a list of files to check for conversion.

pfconv
A pointer to a DFCPFCONVERT structure that contains a list of platform conversion functions. You can pass NULL if you wish to use the default platform conversion functions.

Key
The Btrieve key to be used if a data file conversion is needed. Pass -1 to convert the data file in physical record order.

Direction
The direction to read, if a data file conversion is needed. Valid values are DFC_FORWARD or DFC_BACKWARD.

Returns
TRUE if a conversion took place,
FALSE if a conversion did not take place.

4.5.22. dfcRegisterGDB()

```
GBOOL                               /*  TRUE if conversion took place       */
dfcRegisterGDB(                     /*  Register conversion for GDB         */
const CHAR *descrp,                 /*  Description of conversion           */
const DFCLIST *dfc,                 /*  Pointer to list of information      */
const DFCPFCONVERT *pfconv,         /*  Pointer to OS conversions           */
const CHAR *ModuleName);            /*  Module name registering             */
```

Notifies the data file conversion API of a conversion that takes place in the generic database.

descrp
A 30 character or less description of the data file to check. Anything past 30 characters is truncated.

dfc
A pointer to a DFCLIST structure that contains a list of files to check for conversion.

pfconv
A pointer to a DFCPFCONVERT structure that contains a list of platform conversion functions.

ModuleName
The module name from the generic database. This value is used in determining which records get passed to your conversion functions.

Returns
TRUE if a conversion took place,
FALSE if a conversion did not take place.

4.5.23. DFCCONVERT (Conversion functions)

```
typedef VOID* DFCFUNCTION;
typedef DFCFUNCTION (*pDFCFUNCTION)(pDFCFILEINFO from,pDFCFILEINFO to);
```

These functions are created to handle data conversion from one form to another. They are normally set up as part of a DFCFILE structure. Platform conversion functions handle modifying the latest version of DOS to the latest version of WNT only. If you are converting from an older version of a DOS file to WNT, your file is first converted to the newest version of DOS, then your platform conversion function is called to convert it to WNT.

from
Pointer to a DFCFILEINFO structure containing information about the file being converted.

to
Pointer to a DFCFILEINFO structure containing information about the file converting to.

Returns
This function must return a pointer to the converted data. You can use the to->Data field to store your converted data. If you are converting a record that is variable length, the from->DataLength reflects the size of the converting from record. You must specify the length of the converting to record by modifying to->DataLength. You may return NULL to NOT convert this record.

4.6. Creating a custom conversion program

Every custom conversion program requires that you include the file DFCAPI2.H. This header file contains all function prototypes and structure definitions. The following is a step by step procedure for creating WGSVBCVT.EXE (The file that converts WGSVBL2.DAT).

4.6.1. Create the DFCLIST structure

The Worldgroup Variables data file exists on both DOS and NT/95 platforms. The following structure was created:

```
static DFCLIST VariableList[]={
     {DFC_DOS, VariableDOS},
     {DFC_WNT, VariableWNT},
     {DFC_END, NULL},
};
```

VariableDOS and VariableWNT are pointers to DFCFILE structures.

4.6.2. Create the DFCFILE structures

The DOS versions of the Worldgroup Variables data file are BBSVBL.DAT and WGSVBL2.DAT. Thus, the following was created:

```
static DFCFILE VariableDOS[]={
     {"BBSVBL",     NULL,               bbsvbl},
     {"WGSVBL2",    ConversionDOS1,     wgsvbl2},
     {"", NULL, NULL},
};
```

Both bbsvbl and wgsvbl2 are pointers to DFCDATAFIELDS structures that define the contents of each data file. ConversionDOS1 is a custom conversion function to convert BBSVBL.DAT to WGSVBL2.DAT.

There is only one version of the Worldgroup Variables data file for NT/95, Thus:

```
static DFCFILE VariableWNT[]={
     {"WGSVBL2",    NULL,               wgsvbl2},
     {"", NULL, NULL},
};
```

The reason wgsvbl2 is contained within both file lists is because each file has the exact same data field layout. The only difference is the PACKED vs. UNPACKED.

4.6.3. Create the DFCDATAFIELDS structures

As seen in the creation of the DFCFILE structures, that are two different DFCDATAFIELDS structures that need to be created:

```
static DFCDATAFIELDS bbsvbl[]={
     {"KEY",         DFCFLD_CHAR,    4,              NULL},
     {"VARSTUFF",    DFCFLD_VCHAR,  OVBRECSZ-4,     NULL},
     {"", DFCFLD_END, NULL},
};
```

```
static DFCDATAFIELDS wgsvbl2[]={
     {"KEY",         DFCFLD_CHAR,   4,                         NULL},
     {"VARSTUFF",    DFCFLD_VCHAR,  sizeof(struct sysvb3)-4, NULL},
     {"", DFCFLD_END, NULL},
};
```

Notice the DFCFLD_VCHAR fields (Variable CHAR) specify the largest record size that "VARSTUFF" could possibly be.

4.6.4. Creating the platform conversion function list

On most custom conversion programs there is no need for a special function to convert data from the DOS platform to the WNT platform. However, for WGSVBCVT.EXE, there was a need to modify the data manually before saving the WNT version. When this need arises, it is necessary to create a DFCPFCONVERT structure that contains a list of platform conversion functions:

```
static DFCPFCONVERT PlatformConverts[]={
     {DFC_DOS, DFC_WNT,  PlatformConvert},
     {DFC_END, DFC_END, NULL},
};
```

The function PlatformConvert will be called when converting data from DOS to WNT.

4.6.5. Create the custom conversion functions

As part of the DFCFILE structures, we specified one conversion function: ConversionDOS1(), and as part of the DFCPFCONVERT structure, we also specified on conversion function: PlatformConvert(). Custom conversion functions are created when you need to transform one version of a datafile to another version, and the transformation requires that you add new fields, or re-adjust somethings value. Custom conversion functions are passed two pointers to DFCFILEINFO structures which provide a wealth of information about both the file you are converting from and the file you are converting to. Here are the two custom conversion functions for WGSVBCVT.EXE:

```
static DFCFUNCTION                /*  Pointer to data                     */
ConversionDOS1(                   /* Converstion from BBSVBL to WGSVBL2   */
pDFCFILEINFO from,                /*  Pointer to from data               */
pDFCFILEINFO to)                  /*  Pointer to to data                 */
{
     return(ConversionVBL(from,to,TRUE,FALSE,NGROUPS));
}

static DFCFUNCTION                /*  Pointer to data                     */
PlatformConvert(                  /* Converstion from BBSVBL to WGSVBL2   */
pDFCFILEINFO from,                /*  Pointer to from data               */
pDFCFILEINFO to)                  /*  Pointer to to data                 */
{
     return(ConversionVBL(from,to,FALSE,TRUE,NUMGRPS));
}
```

The ConversionVBL() function handles all possible data conversions for WGSVBCVT.EXE, and takes the following paramaters:

```
static DFCFUNCTION                /*  Pointer to data                     */
ConversionVBL(                    /* Converstion function                 */
pDFCFILEINFO from,                /*  Pointer to from data                */
pDFCFILEINFO to,                  /*  Pointer to to data                  */
GBOOL EatOldCreds,                /*  Eat old credits up?                 */
GBOOL TakeLongs,                  /*  Take variables as longs? (or shorts)*/
INT nGrp)                         /*  Number of groups                    */
```

from
Pointer to DFCFILEINFO structure received in conversion function.

to
Pointer to DFCFILEINFO structure received in conversion function.

EatOldCreds
From Worldgroup 2.00 to Worldgroup 3.00, there were a number of credits tracking variables in this structure that were removed. Specifying TRUE removes those variables.

TakeLongs
From Worldgroup 2.00 to Worldgroup 3.00, there were a number of variables that were changed from SHORTs to LONGs. Specifying TRUE expects the data to be of type LONG.

nGrp
The number of groups we track for changed from Worldgroup 2.00 to Worldgroup 3.00.

4.6.6. Putting it all together

Now that every structure required for the conversion is complete, there is only one thing left: registering all of this information with the data file conversion API. Your programs main() routine should look like this:

```
INT
main(VOID)
{
TRY
     dfcRegister("Worldgroup Variables",VariableList,PlatformConverts,
      -1,DFC_FORWARD);
EXCEPT
     return(0);
}
```

Your conversion programs no longer receive command line arguments to specify what is being converted. Instead, it is the conversion APIs job to figure out what needs to be converted based on the structures you have created. Calling dfcRegister() notifies the conversion API of the different files, datatypes and conversion functions, and everything else is handled for you. Your conversion functions are only called if a data file conversion takes place.

4.6.7. Adding another data file version

This is the easy part. If the need ever arises to create another version of a data file that already has a custom conversion, it is simple to add it to your conversion program. For instance, say we created a new version of Worldgroup Variables that contained a LONG variable in between the “key” and “varstuff” variables. We would have to do the following (additions are in bold and italicized):

1. Add it to our files list, they would now look like this:

```
static DFCFILE VariableDOS[]={
     {"BBSVBL",     NULL,               bbsvbl},
```

```
     {"WGSVBL2",    ConversionDOS1,     wgsvbl2},
     {"WGSVBL3",    ConvertTo3,         wgsvbl2},
     {"", NULL, NULL},
};

static DFCFILE VariableWNT[]={
     {"WGSVBL2",    NULL,               wgsvbl2},
     {"WGSVBL3",    ConvertTo3,         wgsvbl3},
     {"", NULL, NULL},
};
```

2. Add our new data fields**:**

```
static DFCDATAFIELDS wgsvbl3[]={
     {"KEY",         DFCFLD_CHAR,   4,                          NULL},
     {"NEWVAR",      DFCFLD_LONG,   1,                          NULL},
     {"VARSTUFF",    DFCFLD_VCHAR,  sizeof(struct sysvb3)-4, NULL},
     {"", DFCFLD_END, NULL},
};
```

3. Create the function ConvertTo3:

```
static DFCCONVERT
ConvertTo3(
pDFCFILEINFO from,
pDFCFILEINFO to)
{
       UINT NewvarOffset;
       pDFCOFFSET pnewvar;
       LONG NewvarSet=10;
       DFCCONVERT Answer;

       // Get the offset in the new data of our new variable
       pnewvar=dfcFindOffset(to,"NEWVAR");
       NewvarOffset=pnewvar->Offset;

       // We can use the default conversion function because nothing is changing
       //  except the fact that we are adding a new variable
       Answer=dfcConvertDefault(from,to);

       //  Add the new variable to the data
       memcpy(&Answer[NewvarOffset],&NewvarSet,sizeof(LONG));

       //  return the new data.
       return(Answer);
}
```

4. Remove the Platform conversion functions:

Because our new data file does not require any special attention when going from DOS to WNT, we no longer need a platform conversion function. We need to remove our DFCPFCONVERT structure, and change our dfcRegister line to the following:

```
dfcRegister("Worldgroup Variables",VariableList,NULL,-1,DFC_FORWARD);
```

5. InstallShield V5.1

5.1. New Features

5.1.1. Uninstall

Now any program installed with InstallShield will have the ability to uninstall itself. This was not implemented in the previous version given to ISVs.

5.1.2. Registry

The registry was 100% re-done, following Microsofts guide lines. "\\SOFTWARE*Company**Product*\\CurrentVersion" will now be where all registry options shall be put, this includes add-ons. The install program will maintain serial number (activation code, if used will be under products registry), and directories (under server directory). It will also pay attention to version numbers of product and server. If installing an Add-on for the Server version 3.12 and Server version 4.0 is installed it will inform the user that the software being installed is for an older version of the software. If installing an Add-on version 4 and Add-on version 5 is already installed it will again notify the user.

5.1.3. Media

You now have the ability to create "Disk", "CD", "FromTheWeb" installs.

5.1.4. Modifiable MSG Files

You can now ask users MSG file options and they can answer that as part of the installation program. The install program will ask the question just as it is in the MSG file, *note* if there is no help, the install program will bomb complaining about it! There is an option to automatically force MSG options to be set, increased, or decreased as the program needs. The user will not see anything when forcing message files to be set. The option will be saved in the registry, so that it will not let the option be modified twice.

5.1.5. Run Executable

The ability to open up a box showing a message like "Merging files" can be done with show message, followed by running an executable to do the merging and then calling a hide message.

5.1.6. Activation Codes

The ability to place activation codes in MSG files is done directly from the install program.

5.1.7. Folders

Any installation program may now insert links into a folder group. When reading the options for "ASK PATH", if it finds the option "Folder=", the install program will read the file and add or remove items from the selected folder group.

5.1.8. INI file

The ini file has been enhanced quite a bit. Unfortunately InstallShield has done away with their command line building utilities for the moment, they have informed us that they will be releasing the next product with them. Before loading InstallShield5 the ini file will need to be placed in the appropriate directory, or after opening InstallShield5 edited via the GUI before attempting to create a build.

5.2. Requirements

InstallShield Professional Version 5.1 must be purchased by each ISV.

InstallFromTheWeb is required if you want to allow anyone to download your software in pieces or install directly from the web.

5.3. Example

***NOTE* - a semi-colon denotes that the option is not going to be used. Placing the semi-colon in front of any option will cause the install program to ignore that option.**

[ChangeOption]
;File=WGSMAJOR.MSG
;Option1=GROUP1
;Adjust1=SPX
;Registry1=WG310

- File - File to open.
- Option# - Message option to change.
- Adjust# - Sets Text, Binary, and String options or increases/decreases Numbers, Longs, and Hex options.
- Registry# - Used for upgrading purposes (with this you never have to remove the option, makes upgrading easier).

[Install For]
Company=Galacticomm
Product=Worldgroup Server
VersionMajor=3
VersionMinor=10

- [Install For] is a requirement and explains which program it will be installing for.
- Company will be used with the registry to retrieve where the product was installed. Must always be "Galacticomm".
- Products will also be used with the registry, and a few questions. Currently we are using "Worldgroup Server", "WebCast ProServer", and "ActiBase Server". Please note that the registry is case sensitive, so these must be placed in exactly as shown.
- VersionMajor will contain the major revision number of the product, in hex, in this case it shall be "3" for Worldgroup Server, "2" for WebCast ProServer, and

"1" for ActiBase Server.

- VersionMinor will be the minor revision number of the product, in hex, in this case "10" for Worldgroup Server means ".10", "2" for WebCast ProServer means ".02", and "0" for ActiBase Server means ".00".

[Install Information]
Company=Galacticomm
Product=Lan Option
VersionMajor=3
VersionMinor=10
DisplayName=Lan Option
DisplayVersion=v3.12
SETHKEY=HKEY_LOCAL_MACHINE
Publisher=Galacticomm, Inc.
OperationsSupport=0115
HelpTelephone=(954) 583-5990
URLUpdateInfo=http://www.gcomm.com/updates.htm
URLInfoAbout=http://www.gcomm.com
CheckServer=TRUE
;CheckODBC=
;FinishApp=
;FinishAppCommandLine=
;HelpLink=

- [Install Information] is a requirement and explains which program it will be installing.
- Company will be used with the registry to retrieve where the product was installed, this will be your company.
- Products will also be used with the registry, and a few questions. Please note that the registry is case sensitive, so be sure to put the exact case in each time.
- VersionMajor will be the major revision number of your product.
- VersionMinor will be the minor revision number of your product.
- DisplayName is not a requirement, and is not directly used by Galacticomm.
- DisplayVersion is not a requirement, and is not directly used by Galacticomm.
- SETHKEY is a requirement that will advise the install program which registry it should use. Normally under add-on installations you will be using "HKEY_LOCAL_MACHINE", if you are doing something that is user related you would use "HKEY_CURRENT_USER".
- Publisher is not a requirement, and is not directly used by Galacticomm.
- OperationsSupport is not a requirement, and is not directly used by Galacticomm. This is noted as a build number.
- HelpTelephone is not a requirement, and is not directly used by Galacticomm. This option is most notably used in about screens of offline utilities.
- URLUpdateInfo is not a requirement, and is not directly used by Galacticomm. This option is most notably used in about screens of offline utilities. This should also load up a browser if clicked.

- URLInfoAbout is not a requirement, and is not directly used by Galacticomm. This option is most notably used in about screens of offline utilities. This should also load up a browser if clicked.
- CheckServer will make sure the server and offline utilities are not running before beginning its install, and will also make calls to run WGSINT.
- CheckODBC makes sure ODBC version 3.0 is installed.
- FinishApp will launch a program at the end of the install program.
- FinishAppCommandLine is its command line.
- HelpLink is a link to the help file.

[Initialize Installation]
BitmapFile=GCOMM.BMP
BitmapPlace=LOWER_LEFT
BitmapHDist=50
BitmapVDist=50
Type=WELCOME
Next=License File
;NoTitle=
;TitleText=
;TitleSize=
;TitleColor=

- [Initialize Installation] is a requirement and begins the install program.
- BitmapFile is not required, but is the name of a file you would like shown on the install screen.
- BitmapPlace is not required, but can be set to LOWER_LEFT, LOWER_RIGHT, UPPER_LEFT, UPPER_RIGHT, and CENTERED. The install will default to CENTERED.
- BitmapHDist is not required, but is the horizontal distance from the corner chosen. This option can not be used with CENTERED.
- BitmapVDist is not required, but is the vertical distance from the corner chosen. This option can not be used with CENTERED.
- Type is a required field, there are currently eleven (11) options to choose from, DONE, WELCOME, WELCOME BMP, PICK OPTION, ASK PATH, SHOW MESSAGE, LICENSE, REGISTER, HIDE MESSAGE, RUN APP, and MESSAGE OPTION. (see definitions for explanations)
- Next is a required field. This will bring you from [Initialize Installation] to [License File], or what ever option is specified.
- NoTitle is for those who wish not to have any text on the main screen.
- TitleText is for those who wish to change the text from something other than "*Product* Installation"
- TitleSize is for those who wish to change it from 36 pitch font.
- TitleColor is for those who want something other than WHITE to show up.

[License File]

Type=LICENSE
LicenseFile=Gcomm Add-On.Lic
Next=Register

[Register Program]
Type=REGISTER
MessageLine1=Please enter the activation code for the "Lan Option".
Next=Ask for Path

[Ask for Path]
Type=ASK PATH
MessageLine1=Please enter the location for Worldgroup Server. Lan Option will install itself, and other needed components.
Reference=GALSPX.DLL
Next=Install Lan Channels

[Install Lan Channels]
Type=PICK OPTION
Option1=Channel Grouping #1
Option2=Channel Grouping #2
Option3=Channel Grouping #3
Next=Channel Grouping #1

[Channel Grouping #1]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=GROUP1
Next=Starting Channel #1

[Starting Channel #1]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=START1
Next=Number of Channels #1

[Number of Channels #1]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=NUMBR1
Next=Server Name #1

[Server Name #1]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=SAPNAM1
Next=Finish Script

[Channel Grouping #2]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=GROUP2
Next=Starting Channel #2

[Starting Channel #2]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=START2
Next=Number of Channels #2

[Number of Channels #2]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=NUMBR2
Next=Server Name #2

[Server Name #2]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=SAPNAM2
Next=Finish Script

[Channel Grouping #3]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=GROUP3
Next=Starting Channel #3

[Starting Channel #3]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=START3
Next=Number of Channels #3

[Number of Channels #3]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG
Option=NUMBR3
Next=Server Name #3

[Server Name #3]
Type=MESSAGE OPTION
File=WGSMAJOR.MSG

Option=SAPNAM3
Next=Finish Script

[Finish Script]
Type=DONE
FirstMessage1=Setup has finished installing the Lan Option. Please read the release notes for additional information.
ReadMe=LAN.RLN

***Type* Definitions:**

ASK PATH
- **Title** - Enter the title of the dialog box (optional)
- **default** - Location of REGPRODUCT
- **MessageLine#** - Enter the Beginning Message for the dialog box (optional)
- **default** - Set By InstallShield
- **Reference** - File that will be referenced for uninstall
- **FolderFile** - File that will be referenced for adding folders (optional)
- **RegistryFile** - File that will be referenced for adding registry options (optional)
- **RenameFile** - File that will be referenced for renaming files (optional)
- **CopyFile** - File that will be referenced for copying files to a temporary DIR, then back after installing, as not to overwrite files in the directory (optional)
- **DeleteFile** - File that will be referenced for deleting files (optional)

DONE
- **Title** - Enter the title of the dialog box (optional)
- **default** - Finished
- **FirstMessage#** - Enter the Beginning Message for the dialog box (optional)
- **default** - Set By InstallShield
- **LastMessage#** - Enter the Ending Message for the dialog box (optional)
- **default** - Set By InstallShield
- **ReadMe** - Enter a read me file or release notes for product (optional)
- **RunMe** - Enter an optional program a user may select here (optional)
- **CommandLine** - Command line if needed by RunMe (optional)

HIDE MESSAGE
Removes Message Box
-- See **SHOW MESSAGE**

LICENSE
- **Title** - Enter the title of the dialog box (optional)
- **default** - *Product* License Agreement
- **FirstMessage#** - Enter the Beginning Message for the dialog box (optional)
- **default** - Set by InstallShield
- **LastMessage#** - Enter the Ending Message for the dialog box (optional)
- **default** - Set by InstallShield
- **LicenseFile** - Enter the license file to be included with the project

MESSAGE OPTION

- **File** - File used to retrieve message option from
- **Option** - Option to get and set

PICK OPTION

- **Title** - Enter the title of the dialog box (optional)
- **default** - Pick an option
- **Message#** - Enter the Message for the dialog box (optional)
- **default** - Please select the option you wish to modify
- **Option#** - Enter the options you wish for the user to select

REGISTER

- **Title** - Enter the title of the dialog box (optional)
- **default** - Register *Product*
- **MessageLine#** -Enter the Beginning Message for the dialog box (optional)
- **default** - Set by InstallShield

RUN APP

- **RunApp** - Enter the application to run
- **Directory** - Enter the directory the application will be in
- **TARGETDIR** - Install Directory
- **WINDIR** - Windows Directory
- **WINSYSDIR** - Windows System Directory
- **SRCDIR** - Install Source Directory
- **SUPPORTDIR** - Temporary Directory
- **default** - **TARGETDIR**
- **CommandLine** - Enter the command line for RunApp (optional)
- **Wait** - If you wish the program not return access until it has finished insert **WAIT=TRUE**. This will pause InstallShield till the task is complete

SHOW MESSAGE

- **FirstMessage#** - Enter the Message to be displayed
- -- This is normally before running a program that doesn't show any information

WELCOME

- **Title** - Enter the title of the dialog box (optional)
- **default** - Welcome
- **MessageLine#** - Enter the Beginning Message for the dialog box (optional)
- **default** - Set by InstallShield

WELCOME BMP

- **Title** - Enter the title of the dialog box (optional)
- **default** - Welcome

- WelcomeBmp **- BMP file to show**

6. Year 2000 issues

6.1. Worldgroup Baseline Changes Required For Year 2000 Compliance

The year 2000 issue has at its heart one problem: the use of two digits to represent a year value that requires more than two digits for a complete representation. For example, it is common practice when writing a date to represent 1998 as 98. This practice was also adopted by many computer programs and can cause bugs when the century portion of the year can not be inferred with 100% accuracy or when dates are sorted using a two-digit representation of the year.

What is known as "year 2000 compliance" is defined differently by different organizations. However, all definitions come down to one issue: will the software continue to work correctly after the year 2000? In Worldgroup, most dates are stored using DOS binary format which is valid until 2108, or VB binary format which is valid until 9999. However, three major areas were identified that were not year 2000 compliant and may affect developers: the birth date field of the user account structure, the date format used by the Audit Trail, and certain date entry functions used by client applications.

6.1.1. User Account Birth Date

The user account birth date (usually referenced through usaptr->birthd) was stored as a string using the U.S. standard month/day/year format, with two digits used to represent the year. This may not have been a serious problem -- after all, the likelihood of having users on a system who were born both in 1900 and in 2000 at the same time is rather small. Even so, it is not impossible, and age discrimination is usually considered undesirable. However, the problem was greater than that because previous versions of Worldgroup will always consider the century to be 1900 when calculating age.

To fix this problem, the format of the birth date field was changed to a string containing year, month, and day packed together in that order. For example, March 24, 1978 is now represented by the string "19780324". Since this format requires the same amount of storage space as the old format, no data file conversion is required. The software will automatically re-format the birth date when the account is loaded (assuming 1900 as the century if the date is in the old format). In addition, all data entry fields used for the birth date have been lengthened to accommodate a four-digit year.

6.1.2. Audit Trail

In previous versions of Worldgroup, Audit Trail information was stored in a Btrieve data file. The time stamp on each entry was stored as a string with a two-digit year. In addition, the time stamp was used as the key for the data file. Thus, when the year 2000 arrived, the Audit Trail would start adding entries to the beginning of the list rather than at the end (after all, "00" is less than "99").

This problem was fixed by creating an entirely new file format. Since Audit Trail information is usually only appended, and random access to the data is usually not a high priority, the Audit Trail is now stored in a binary/text file instead of a Btrieve data file. The exact nature of the new storage format is discussed below and in the Audit Trail API documentation.

6.1.3. Client Application Date Entry

Galacticomm's client applications have historically used a masked edit box for date entry. The standard format used for the masked edit box was two digits for the month, two digits for the day, and two digits for the year. The two-digit limitation on years is clearly not in keeping with year 2000 compliance. Since this limitation had to be removed, and since there are standard functions in VB that will recognize any valid date string, all masked edit boxes being used for date entry were replaced with standard text boxes. Users can now enter any valid date and it will be recognized by the clients.

6.2. API Changes And Additions

6.2.1. Birth Date Manipulation Functions

Fortunately, all Galacticomm modules use the same function to validate birthdates:

```
INT                                 /*   returns TRUE if valid             */
okbday(                             /* is this a valid birthday?           */
CHAR *stg);                         /*   date str buffer (must be DATSIZ)  */
```

This function can be found in SIGNUP.C. It has been modified to do the following:

1. Accept dates in the format MM/DD/YY, MM/DD/YYYY, MMDDYY, or MMDDYYYY. If two digits are used for the year, 1900 is assumed for the century.

2. Modify the input to contain the new YYYYMMDD format.

The current age of a user can be calculated using the following function:

```
INT
calcage(                            /* compute age (in years) as of today  */
const CHAR *birthd);                /*   birth date (YYYYMMDD or MM/DD/YY) */
```

This function also existed in earlier versions of Worldgroup, but it has been modified to accept either the new or the old date format (YYYYMMDD or MM/DD/YY). Again, 1900 is assumed for the century if the date is in the old format.

Two new functions have also been added specifically for dealing with user birth dates:

```
GBOOL                               /*   returns TRUE if modified          */
fixBirthdate(                       /* fix date from MM/DD/YY to YYYYMMDD  */
CHAR *strDate);                     /*   birth date string (must be DATSIZ)*/

const CHAR *                        /*   returns pointer to static buffer  */
strBirthdate(                       /* format birthdate into "MM/DD/YYYY"  */
const CHAR *strDate);               /*   birth date string                 */
```

The first function is specifically for converting old format dates to the new format. It examines the input and converts it to the new date format if it is in the old format. It does not accept four-digit years in the old format and it always assumes 1900 as the century. This function is called each time the user account is loaded to ensure that the birth date is in the correct format.

The second function is used to convert the birth date string (in either the old or new format) to a version that can be used for displays. The output is the date in U.S. standard format, MM/DD/YYYY.

6.2.2. Date and Time API

To facilitate year 2000 compliance, a number of new functions have been added to the existing date and time API and some existing functions have been modified. These changes fall into three basic categories: changes to support the use of four-digit years, new functions to support a "sort" date format, and new functions to support dates treated as discrete components. Complete details on these functions can be found in the Date and Time API documentation. An overview of the important changes follows.

Four-Digit Years:
Four-digit years obviously require more storage space than two-digit years. There is a global define that specifies the amount of space required to store a date string in MM/DD/YY format:

```
#define DATSIZ 9                    /* size of a date in MM/DD/YY format   */
```

This define was originally in USRACC.H. It has been moved to DNTAPI.H. To accommodate dates with four-digit years, a new define was created:

```
#define LDATSIZ 11                  /* size of a date in MM/DD/YYYY format  */
```

The primary function for decoding a string into a DOS binary date is dcdate(). This function was already reasonably well-behaved. It accepted only two-digit years, but it would assume years less-than 80 were in the 2000s and years greater-than or equal-to 80 were in the 1900s. This functionality has been retained, but it has also been modified to accept four-digit years.

The primary functions for creating a date string from a DOS or VB binary date are ncdate(), ncedat(), and v2sdat(). These functions all create date strings in with two-digit years. New, "long", versions of these functions have been added that create date strings with four-digit years: ncdatel(), ncedatl(), and v2sdatl().

Sort Format Date Strings:
In order to provide a convenient, compact format that contains complete date information and can be used to easily sort dates and times, new date and time string formats have been defined: the sort formats.

The sort date format is the format now used for birth dates in the user account structure. It consists of a four-digit year, immediately followed by a two-digit month, immediately followed by a two-digit day. For example, January 2, 1980 is represented by "19800102".

The sort time format is similar to the existing time string format, but contains no punctuation. It consists of a two-digit hour (in 24-hour format), immediately followed by a two-digit minute, immediately followed by a two-digit second. For example, three seconds before midnight (11:59:57 PM) is represented by "235957".

A suite of functions for encoding and decoding sort date and time strings in various ways has been added. These functions are described in detail in the Date and Time API documentation. They can be identified easily because the function names all begin with a lower-case s.

Discrete Components Binary Format:
The most flexible binary date format is using a separate integer variable to hold each of the year, month, and day components of a date. This is referred to in the documentation as the "discrete components" format. A suite of functions has been added to manipulate discrete component dates. Most of these functions mirror functions that existed for manipulating DOS packed dates. See the Date and Time API documentation for details.

6.2.3. Audit Trail API

The Audit Trail data file format was changed for Worldgroup 3.12 for two main reasons: to provide year 2000 compliance, and to provide easier access to the Audit Trail data.

Historically, the Audit Trail has been stored in a Btrieve data file. This provided a standard access mechanism, since most data in Worldgroup is stored in Btrieve files, and under the DOS versions of Worldgroup, this provided a certain amount of data security, since the Btrieve engine has built-in mechanisms to ensure file integrity. The primary down-side of storing any information in a Btrieve file is that the data is difficult for non-Worldgroup products to access the data. In most cases, custom programming is required to read the data from the Btrieve file and convert it to a more generic format that can be read by general-purpose data manipulation utilities.

For Worldgroup 3.1, Audit Trail data is stored in a binary/text file. The file is opened and accessed in binary mode by Worldgroup itself, but the contents of the file are formatted so that it can be treated by outside programs as a fixed-length ASCII data file. See the Audit Trail API documentation for details.

Because of this change, from a Btrieve file to a special type of file, some changes to add-on modules may be required. If you only add items to the Audit Trail using shocst(), no changes will be required. This function still operates exactly as it did before. In addition, all hooks called by shocst() (ntfysopr and rsyfad_hook) operate exactly as before.

Add-ons that read from the Audit Trail will require changes, however. The Audit Trail API provides generic access to Audit Trail files. When reading from the Worldgroup Audit Trail, always use the global handle which is opened by the baseline. This handle is declared in MAJORBBS.H and imported through WGSERVER.LIB:

```
HAUDFILE WGSEXP hAuditTrail;       /* Audit Trail file handle             */
```

Offline utilities that operate only while the Worldgroup Server is not running can open the file themselves. There is a define in MAJORBBS.H that can be used to get the name of the file:

```
#define AUDITFILE "wgsaudit.adt"   /* Audit Trail file name               */
```

See the Audit Trail API documentation for details on opening and closing Audit Trail files.

Audit Trail files are treated by the API as binary files containing records. The records are numbered from zero to the one less than the total number of records. You can get the number of records by calling the following function:

```
ULONG
audfNumRecs(                       /* get number of records in file       */
HAUDFILE haud);                    /*   Audit Trail file to use           */
```

The format of records in the new Audit Trail is very similar to that in the old one. The main differences are the format of the time stamp and that the maximum length of the detailed information has been increased. The new format has also been formalized using the following structure:

```
struct audEntry {                  /* Audit Trail record format           */
     CHAR stamp[AUDSTAMPSIZ];      /*   time stamp ("YYYYMMDD HHMMSS")    */
     CHAR brief[AUDBRIEFSIZ];      /*   brief description                 */
     CHAR channel[AUDCHANSIZ];     /*   channel/console/event/cleanup     */
     CHAR detail[AUDDETSIZ];       /*   detailed description              */
};
```

The main function to read a single record by number is:

```
INT                                /*   returns result code               */
audfReadEntry(                     /* read an entry by record number      */
HAUDFILE haud,                     /*   Audit Trail file to use           */
ULONG recNum,                      /*   record number to read             */
struct audEntry *buf,              /*   buffer to receive data            */
size_t bufSiz);                    /*   size of destination buffer        */
```

If you have a large amount of existing code that deals with the old Audit Trail record format and you don't want to change it all to work with the new format, you can use the following function instead:

```
INT                                /*   returns result code               */
audfReadOldStyle(                  /* read an old-style entry             */
HAUDFILE haud,                     /*   Audit Trail file to use           */
ULONG recNum,                      /*   record number to read             */
struct audOldStyle *buf,           /*   buffer to receive data            */
size_t bufSiz);                    /*   size of destination buffer        */
```

The audOldStyle structure is simply a formalization of the old record format. You can pass a character

buffer and simply typecast it when using this function.

There are also functions to find records by date. See the Audit Trail API documentation for details on these functions.

6.3. Compliance Guidelines

6.3.1. Handling Birth Date

Online modules that accept birth dates as input should use okbday() to validate the input. The user input should be passed to this function and the modified date string should then be copied into the user account structure.

Modules that read the birth date may need to be able to handle both formats. Modules written only for Worldgroup version 3.12 and later should call fixBirthdate() before using the birth date field, especially if the account record is read directly from the data file. Then the birth date can be decoded using sDateDecode(). Modules that must remain compatible with Worldgroup version 3.0 can use the GCOMM library function alldgs() to determine whether a birth date is in the new format or the old format. For example:

```
INT year,month,day;

if (alldgs(usaptr->birthd)) {
     /* it's the new format */
     sscanf(usaptr->birthd,"%4d%2d%2d",&year,&month,&day);
}
else {
     /* it's the old format */
     sscanf(usaptr->birthd,"%d/%d/%d",&month,&day,&year);
}
```

6.3.2. Using the New Audit Trail

Complete details on using the new Audit Trail are given in the above section on API Changes and Additions and in the Audit Trail API documentation. The key points to keep in mind are:

- Discontinue use of audbb and the WGSAUDT2.DAT file.

- Online modules should read from the Audit Trail using hAuditTrail.

- Offline utilities should get the Audit Trail file name from the AUDITFILE define in MAJORBBS.H.

6.3.3. Date Output

On the server, dates that are output for display only and for which the century can be implied with a high degree of accuracy do not strictly need to be changed to use four-digit years. However, you should be aware that some organizations require a complete year in all date displays in order to consider a product "year 2000 compliant".

Dates that are output for data entry purposes should in general be changed to use four-digit years. If you are using prndat() for date output, you can change the year to four digits simply by adding 12 to the mode argument. If you are using ncdate() or ncedat(), change these calls to use ncdatel() or ncedatl(), respectively.

All client-side date output should now be done using the standard VB library function Format$(). In particular, you should change all calls to the Galacticomm function mebdate() to use Format$() instead. In addition to the existing standard date format strings defined in GCSPCLI.BAS (DATEFMT, MDATEFMT, LDATEFMT, and SERVDFMT), two new format strings have been added. The first one, LSRVDFMT, is

equivalent to the existing SERVDFMT format. It generates a date string in the format MM/DD/YYYY which can be decoded using dcdate(). The difference between the two is that SERVDFMT generated a string with a two-digit year while LSRVDFMT generates a string with a four-digit year. The second new format string, SORTDFMT, is used to generate a sort format date string (YYYYMMDD).

6.3.4. Date Input

On the server, change the size of date string buffers to LDATSIZ instead of DATSIZ. For FSD templates, increase the length of date entry fields from eight to ten. You can still use dcdate() for decoding date strings to DOS packed format or you can use the new dateDecode() function to decode to discrete components.

On the client, use a regular text box instead of a masked edit box for date entry fields. When decoding the date string to VB binary format, use the standard VB library function DateValue().

7. Searchable Memory Blocks

With the creation of WorldLink it became obvious that we needed a mechanism for storing large amounts of information on DOS systems. The realization that we could have hundreds, or even thousands of simultaneous users intermingling prompted us to create the Searchable Memory Block API or (GALSMB).

SMBs (searchable memory block) are similar to Btrieve, however, the data is not persistent. You create an SMB by calling smbOpen(), then you can insert, remove or retrieve records on an as needed basis. You specify the number of records to store in memory, and the rest are swapped out to file. Once the server is shut down, all SMBs are closed and the data is lost.

7.1. SMB API

7.1.1. SMBKEYTABLE structure

```
typedef struct smbKeyTable {       /* Key table structure for datafile     */
     USHORT Offset;                /*  Offset of where the key starts      */
     smbCompFunc Compare;          /*  Key comparison routine              */
} SMBKEYTABLE, *pSMBKEYTABLE;
```

Defines the the location, and the routines used to compare keys within the data being inserted into SMBs.

Offset
The offset, within inserted data, that the key starts.

Compare
A pointer to a compare routine. You may use any of the default compare routines, or create your own.

7.1.2. smbClose()

```
VOID
smbClose(                          /*  Close an SMBHANDLE                  */
SMBHANDLE smb);                    /*  Pointer to file to close            */
```

Closes an opened SMB. This function deletes any swap files used, as well as freeing all allocated memory dealing with the opened SMB.

smb
Handle of the SMB returned from smbOpen.

7.1.3. smbCurrentData()

```
VOID *                             /* Pointer to data or NULL              */
smbCurrentData(                    /*  Gets data of current record         */
SMBHANDLE smb);                    /*  Pointer to smb information          */
```

Retrieves a pointer to the last record obtained using the smbGet... functions.

smb
Handle of the SMB to get the current record from.

Returns
Pointer to the current SMB record.

7.1.4. smbCurrentNumber()

```
SMBPTR                             /* Absolute number of current record    */
```

```
smbCurrentNumber(                   /*  Gets current position of record     */
SMBHANDLE smb);                     /*  Handle to smb information           */
```

Retrieves a number associated with the current SMB record. This is similar to the dfaAbs() function. This number can be used in cycled calls to smb data.

smb
Handle of the SMB to get the current record number of.

Returns
A number associated with the current record.

7.1.5. smbDelete()

```
VOID
smbDelete(                          /*  Deletes the current record pointer  */
SMBHANDLE smb);                     /*  Pointer to file to delete           */
```

Deletes the last record read by using one of the smbGet... functions.

smb
Handle of the SMB to delete from.

7.1.6. smbGetBynum()

```
VOID *                              /* Pointer to data or NULL              */
smbGetBynum(                        /*  Get record by absolute number       */
SMBHANDLE smb,                      /*  Handle to smb information           */
SMBPTR Number,                      /*  Record number to get                */
UINT Key);                          /*  Key number to load as current       */
```

Retrieve a record by record number. The record number can be obtained using smbCurrentNumber().

smb
Handle of the SMB to get the record from.

Number
The record number of the record to retrieve.

Key
The key number to load the record by.

Returns
A pointer to the SMB record or NULL if not found.

7.1.7. smbGetEqual()

```
VOID *                              /* Pointer to data or NULL              */
smbGetEqual(                        /*  Get a record from a database        */
SMBHANDLE smb,                      /*  Pointer to file to update           */
const VOID *KeyInfo,                /*  Key we are searching for            */
UINT Key);                          /*  Key number to look up by            */
```

Retrieve a record that has a key equal to the one you provide.

smb
Handle of the SMB to get from.

KeyInfo

Pointer to a key you wish to look up by.

Key
Key Number to compare KeyInfo with. This corresponds to your SMBKEYTABLE structure passed to smbOpen().

Returns
A pointer to the data matching KeyInfo, or NULL if the key was not found.

7.1.8. smbGetGreater() and smbGetLess()

```
VOID *
smbGetGreater(                    /* Get greater than a key              */
SMBHANDLE smb,                    /*  Pointer to file to update          */
const VOID *KeyInfo,              /*  Key we are searching for           */
UINT Key);                        /*  Key number to look up by           */

VOID *
smbGetLess(                       /* Get less than a key                 */
SMBHANDLE smb,                    /*  Pointer to file to update          */
const VOID *KeyInfo,              /*  Key we are searching for           */
UINT Key);                        /*  Key number to look up by           */
```

Retrieve a record greater or less than a specified key value.

smb
Handle of the SMB to get a record from.

KeyInfo
Pointer to a key you wish to look up by.

Key
Key number to retrieve the record by. This corresponds to your SMBKEYTABLE structure passed to smbOpen().

Returns
A pointer to the data matching KeyInfo, or NULL if the key was not found.

7.1.9. smbGetHigh() and smbGetLow()

```
smbGetHigh(                       /*  Get highest record in the tree     */
SMBHANDLE smb,                    /*  Handle to smb information          */
UINT Key);                        /*  Key to get highest on              */

VOID *                            /* Pointer to data or NULL             */
smbGetLow(                        /*  Get the lowest record in the tree  */
SMBHANDLE smb,                    /*  Handle to smb information          */
UINT Key);                        /*  Key to get lowest on               */
```

Get the highest or lowest record by key from an SMB.

smb
Handle of the SMB to get a record from.

Key
Key number to look up with. This corresponds to your SMBKEYTABLE structure passed to smbOpen().

Returns

A pointer to the highest or lowest record.

7.1.10. smbGetPrevious() and smbGetNext()

```
VOID *                          /* Pointer to data or NULL             */
smbGetPrevious(                 /*  Get previous from current record    */
SMBHANDLE smb);                 /*  Handle to current smb pointer       */

VOID *                          /* Pointer to data or NULL             */
smbGetNext(                     /*  Get next from current record       */
SMBHANDLE smb);                 /*  Handle to current smb pointer       */
```

Get the previous or next record in sequence. The key used to retrieve the record is determined by the last call to a get function that requires a key.

smb
Handle of the SMB to get a record from.

Returns
A pointer to the SMB record, or NULL if there are no more records.

7.1.11. smbInit()

```
VOID
smbInit(VOID);                  /* initialize SMB API before using     */
```

Initializes the Searchable Memory Block API. This function must be called before any other SMB APIs are used.

7.1.12. smbInsert()

```
VOID *
smbInsert(                      /*  Insert a record into datafile      */
SMBHANDLE smb,                  /*  Pointer to file to insert to       */
const VOID *data);              /*  Data to insert                     */
```

Inserts a record into an SMB.

smb
Handle of the SMB you want to insert to.

data
A pointer to the data. This data must be the length that you passed to your smbOpen() call.

Returns
A pointer to the newly created SMB record.

7.1.13. smbOpen()

```
SMBHANDLE                       /* Pointer to open file                */
smbOpen(                        /*  Open an in memory file             */
size_t DataLength,              /*  Maximum length of file to open     */
pSMBKEYTABLE keyptr,            /*  Pointer to the key table           */
UINT CachedRecords);            /*  Records to Cache                   */
```

Creates a searchable memory block.

DataLength
Maximum length of the data that will be inserted into this SMB.

keyptr
Pointer to a SMBKEYTABLE structure specifying the different keys that are used during get operations.

CachedRecords
The number of records to be saved in memory at any one time.

Returns
A handle to the newly created SMB.

7.1.14. The Query Routines

The functions smbQueryEqual(), smbQueryPrevious(), smbQueryNext(), smbQueryGreater(), smbQueryLess(), smbQueryLow() and smbQueryHigh() can be used instead of the smbGet... functions. The query set of routines take the same paramaters, but return TRUE or FALSE instead of a pointer to the data.

7.1.15. The default compare functions

The functions smbCompareString(), smbCompareLONG(), smbCompareINT(), and smbCompareSHORT() are generic routines created to compare key types. You may create your own conversion functions to suite your specifc needs. Each function is similar to this one:

```
INT                                 /* LEFTHIGH, EQUAL, or RIGHTHIGH        */
smbCompareString(                   /*  Generic string compare routine      */
const VOID *target,                 /*  target string                       */
const VOID *test);                  /*  string to compare to                */
```

Compares 2 strings and returns which one is greater.

target
Pointer to first string to compare.

test
Pointer to second string to compare.

Returns
LEFTHIGH if target > test,
RIGHTHIGH if test > target,
EQUAL if test = target.

7.2. Compare functions

When records are inserted, it is necessary to compare the inserted key data with existing key data. As records are being inserted into an SMB, a tree is also built for each key. When a new record is inserted, it is continuously compared against the nodes on the tree to find the proper place of insertion. The proper format for a compare routine is as follows:

```
typedef INT                         /* Returns comparison identifier        */\
(*smbCompFunc)(                     /*  Key comparison routine              */\
const VOID *target,                 /*  Pointer to target object            */\
const VOID *test);                  /*  Pointer to object to compare to     */
```

target
Pointer to the first key to compare.

test
Pointer to the second key to compare.

Returns
LEFTHIGH if target > test,
RIGHTHIGH if target < test,
EQUAL if target = test.

A number of default compare routines are provided to aid you in creation of your SMB.

7.3. Sample SMB application

The following is a sample program using the SMB API to store a list of Client/Server users. The users can be listed out by using the global "/USERS".

```
#include "gcomm.h"
#include "smbapi.h"

SMBHANDLE UserSMB;

SMBKEYHANDLE UserKey[]={
       {0,    smbCompareString},
       {0,    NULL},
};

static VOID UsersDown(VOID);
static VOID UserDisconnect(VOID);
static VOID UserConnect(VOID);
static INT GlobalUsers(VOID);

VOID EXPORT
init__userlist(VOID)
{
       smbInit();
       UserSMB=smbOpen(UIDSIZ,UserKey,10);
       hook_shutdown(UsersDown);
       hook_connect(UserConnect);
       hook_disconnect(UserDisconnect);
       globalcmd(GlobalUsers);
}

static VOID
UserConnect(VOID)
{
       smbInsert(UserSMB,usaptr->userid);
}

static VOID
UserDisconnect(VOID)
{
       if (smbQueryEqual(UserSMB,usaptr->userid,0)) {
              smbDelete(UserSMB);
       }
}

static INT
GlobalUsers(VOID)
{
       CHAR *UserData;

       if (margc == 1 && sameas(margv[0],"/USERS")) {
              prf("C/S Users Online:\n");
              if ((UserData=smbGetLow(UserSMB,0)) != NULL) {
                     do {
```

```
                        prf("%s\n",UserData);
                } while ((UserData=smbGetNext(UserSMB)) != NULL);
            }
            return(1);
        }
        return(0);
}

static VOID
UsersDown(VOID)
{
        smbClose(UserSMB);
}
```

8. Audit Trail API

8.1. OVERVIEW

The Worldgroup Audit Trail is used to track system events. A chronological list of these events can by listed by the system operator. Each event consists of a time stamp, a channel or other context description, a brief description of the event, and a detailed description of the event. Worldgroup keeps all of these event records in a single file with a specific format. The Audit Trail API provides a standard mechanism for reading from and writing to the Worldgroup Audit Trail, and for creating other audit trails.

8.1.1. File Format

The files managed by this API are binary/text files. They are treated by the API as binary files with fixed-length records, and referenced by record number. Internally, the files are formatted as fixed-length ASCII text files, so that they can be read by generic data file utilities.

Each record consists of a time stamp, brief description, channel, and detailed description, in that order. Each field is separated from the next by at least one ASCII space character (32 decimal). Fields that are not as long as their maximum length are padded on the right end with ASCII space characters. Each record is equivalent to one line in a text file, thus each record ends with a carriage return and line feed (DOS-format text file).

The fields in each record are as follows:

Field	Offset	Length	Description
Time Stamp	0	15	The date and time at which the entry was added to the file. It consists of a sort format date string, followed by a space, followed by a sort format time string. For example, October 28, 1997 at 4:25:34 PM is represented by: "19971028 162534".
Short Desc.	16	32	A short description of the event; usually all upper case and the same for each similar event.
Channel	49	16	A description of the channel or other context in which the event occurred.
Detail	66	127	A detailed description of the event; usually mixed case and may contain event-specific information.
EOL	194	2	End of line marker ("\r\n").
Total:		**196**	

8.1.2. Online Context

Some audit trail functions require an online context to operate properly. The following global variables are required for the online context: usrnum, nterms, errcod, channel[]. These variables determine what is stored in the channel field of an audit trail record. This is determined as follows:

1. If usrnum is set to one of the special channel identifiers, the following text is used for the channel:

usrnum	Channel Text
AUDCHAN_CONSOLE	Console
AUDCHAN_CLEANUP	Cleanup
AUDCHAN_EVENT	Event #

The event number is calculated by errcod-10.

2. If usrnum is in the range zero to nterms-1, the channel text is "Chan XX" where XX is the hexadecimal value of the channel number as displayed in the channel grid. This number is gotten from channel[usrnum].

3. If usrnum is outside the range zero to nterms-1, the channel text is "U# XXXX" where XXXX is the hexadecimal value of usrnum.

8.1.3. Example

Below is a code example that opens an audit trail file, adds a record, reads and displays the last ten records, and then closes the file.

```
#include "gcomm.h"

/* create online context required for audit trail */
INT usrnum;
INT errcod;
INT nterms=1;
INT channel[1]={0};

INT
main(VOID)
{
     HAUDFILE haud;               /* audit trail file handle             */
     INT rc;                      /* return from audit trail functions   */
     ULONG nRecs;                 /* total number of records in file     */
     ULONG iStart;                /* record at which to start printing   */
     ULONG iRec;                  /* record index for printing           */
     struct audEntry recBuf;      /* buffer for reading audit trail      */
     INT year;                    /* year part of time stamp             */
     INT month;                   /* month part of time stamp            */
     INT day;                     /* day part of time stamp              */
     INT hour;                    /* hour part of time stamp             */
     INT minute;                  /* minute part of time stamp           */
     INT second;                  /* second part of time stamp           */
     CHAR dateBuf[sizeof("September 31, 1999")]; /* buf for date string   */
     CHAR timeBuf[sizeof("11:59:59 PM")]; /* buffer for time string       */

     /* open the audit trail file */
     rc=audfOpen(&haud,"myfile.adt",AUDMODE_READWRITE);

     /* check for error opening */
     if (rc < AUDERR_OK) {
          fprintf(stderr,"Error opening file: %d\n",rc);
          return(1);
     }

     /* check for damaged file */
     if (rc == AUDERR_DAMAGED) {
```

```
     /* attempt to recover */
     rc=audfRecover(haud);

     /* check for error in recovery */
     if (rc < AUDERR_OK) {
          audfClose(haud);
          fprintf(stderr,"Error recovering file: %d\n",rc);
          return(1);
     }
}

/* prepare online context (for Console) */
usrnum=AUDCHAN_CONSOLE;
errcod=0;

/* add a record */
rc=audfAddEntry(haud
               ,"AUDIT TRAIL TEST"
               ,"Testing the new Audit Trail API");

/* check for error */
if (rc < AUDERR_OK) {
     audfClose(haud);
     fprintf(stderr,"Error writing to file: %d\n",rc);
     return(1);
}

/* print the last ten entries */
nRecs=audfNumRecs(haud);
if (nRecs < 10) {
     iStart=0;
}
else {
     iStart=nRecs-10;
}
for (iRec=iStart ; iRec < nRecs ; ++iRec) {

     /* read the current entry */
     rc=audfReadEntry(haud,iRec,&recBuf,sizeof(recBuf));

     /* check for error */
     if (rc < AUDERR_OK) {
          audfClose(haud);
          fprintf(stderr,"Error reading from file: %d\n",rc);
          return(1);
     }

     /* decode the time stamp */
     sDecodeDT(recBuf.stamp,&year,&month,&day,&hour,&minute,&second);
     prnDate(year,month,day,dateBuf,sizeof(dateBuf),PRND_MMMDYY,'\0');
     prnTime(hour,minute,second,timeBuf,sizeof(timeBuf),PRNT_HMS_PMC);

     /* print the entry */
     fprintf(stdout,"Entry #%lu\n",iRec);
     fprintf(stdout,"Time Stamp:   %s  %s\n",dateBuf,timeBuf);
     fprintf(stdout,"Channel:      %s\n",recBuf.channel);
     fprintf(stdout,"Description:  %s\n",recBuf.brief);
     fprintf(stdout,"Details:      %s\n",recBuf.detail);
     fprintf(stdout,"\n");
}

/* close and exit */
audfClose(haud);
```

```
     return(0);
}
```

8.2. Result Codes

Most Audit Trail API functions return a code that describes the result of the operation. In general, a negative code indicates failure while a zero or positive code indicates success. The following is a list of all of the result codes that may be returned by Audit Trail API functions.

Identifier	Value	Description
AUDERR_OK	0	The operation completed successfully.
AUDERR_NEW	1	When opening a file, the specified file was not found, so a new file was created.
AUDERR_DAMAGED	2	When opening a file, it was found to have been damaged, so you must recover the file before proceeding.
AUDERR_FIO	-1	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_MEMORY	-2	There was not enough memory to complete the requested operation.
AUDERR_NOTOPEN	-3	The file is not open, you must close and re-open it before you can use it.
AUDERR_MODE	-4	The requested operation was not valid for the mode used when opening the file (e.g., attempting to write to a read-only file).
AUDERR_NOTFOUND	-5	The requested record was not found (the record number was out of range).

8.3. Structures

The following is a list of structures defined by the Audit Trail API.

```
struct audEntry {                  /* audit trail record format            */
     CHAR stamp[AUDSTAMPSIZ];      /*   time stamp ("YYYYMMDD HHMMSS")     */
     CHAR brief[AUDBRIEFSIZ];      /*   brief description                  */
     CHAR channel[AUDCHANSIZ];     /*   channel/console/event/cleanup      */
     CHAR detail[AUDDETSIZ];       /*   detailed description               */
};

#define AUDSTAMPSIZ sizeof("YYYYMMDD HHMMSS")
#define AUDBRIEFSIZ (32+1)
#define AUDCHANSIZ  (16+1)
#define AUDDETSIZ   128
```

This is the standard structure for reading and writing audit trail records. The fields have the following uses:

stamp: The time and date stamp that describes when the record was added. It is stored in sort date and time format with the date first and a space between the date part and the time part.

brief: A brief description of the event. It is the same for each event of the same type (e.g., "SMTP MAIL RECEIVED").

channel: The channel or event which caused the event. This field usually contains one of the following:

Text	Meaning
Chan XX	The specified channel was being serviced when the event occurred. XX is the channel number in hexadecimal as seen in the channel grid.
U# XXXX	The specified user number was in effect when the event occurred. XXXX is the user number in hexadecimal as represented by the global usrnum. This channel type is used when usrnum is outside the range of defined channels.
Console	The event was generated by the Worldgroup console or by a background task. This channel type is used when usrnum is equal to AUDCHAN_CONSOLE (-1)

Cleanup	The event was generated during cleanup. This channel type is used when usrnum is equal to AUDCHAN_CLEANUP (-2).
Event #	The event was generated while shutting down to perform a scheduled event. # is the event number in decimal. This channel type is used when usrnum is equal to AUDCHAN_EVENT (-3).

detail: Details on the event. This field may contain information specific to the particular event (e.g., "SMTP MAIL RECEIVED", "From: user@company.com To: Local User"). If the information to be placed in this field exceeded the maximum length, the last character before the terminating nul character will be an asterisk ('*').

```
struct audOldStyle {                /* old-style audit trail record format  */
     CHAR stamp[AUDOLDSTAMPSIZ];    /*   time stamp (not nul-terminated)    */
     CHAR brief[AUDOLDBRIEFSIZ];    /*   brief description (32 chars + sp)  */
     CHAR channel[AUDOLDCHANSIZ];   /*   chan/console/event (nul-terminated)*/
     CHAR unused[12];               /*   unused space (filled with nuls)    */
     CHAR detail[AUDOLDDETSIZ];     /*   long desc (63 chars + 2 nuls)      */
};

#define AUDOLDSTAMPSIZ   14
#define AUDOLDBRIEFSIZ   33
#define AUDOLDCHANSIZ    8
#define AUDOLDDETSIZ     65
```

This is a formalized representation of the pre-Worldgroup 3.12 Audit Trail record format. It is defined so that the API can provide backward compatibility with add-ons that read from the Audit Trail. In general, add-ons that use the old format should continue to use the known offsets into a character buffer approach to extracting fields from old-style records rather than using this structure directly.

8.4. Functions

The following is an alphabetical list of all functions provided by the Audit Trail API.

8.4.1. audAddEntry()

```
INT                                 /*   returns result code               */
audfAddEntry(                       /* add an entry to audit trail file     */
HAUDFILE haud,                      /*   audit trail file to use            */
const CHAR *brief,                  /*   brief description                  */
const CHAR *detail,                 /*   detailed description/format string */
...);                               /*   arguments to be formatted          */
```

Adds a record to an audit trail file, with variable arguments. It requires an online context as discussed above.

haud
The file to which to add the record. This handle must have been gotten from a successful call to audfOpen(). The file must have been opened with the mode AUDMODE_READWRITE.

brief
A brief description of the event. This is the same for each event of the same type. It will be truncated at AUDBRIEFSIZ-1 (32) characters.

detail
Details on the event. This is a printf()-style format string which can be used to record event-specific information. It will be truncated at AUDDETSIZ-1 (127) characters. If it is truncated, the last character before the terminating nul character will be an asterisk ('*').

Returns:

A code describing the result of the operation. It may be one of the following:

AUDERR_OK	The operation completed successfully.
AUDERR_NEW	The file was not found, so a new file was created. (This code will not be returned under normal conditions.)
AUDERR_DAMAGED	The file is damaged; you must recover the file before continuing. (This code will not be returned under normal conditions.)
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_MODE	The requested operation was not valid for the mode used when opening the file (e.g., attempting to write to a read-only file).

8.4.2. audAddLowLevel()

```
INT                                 /*   returns result code              */
audfAddLowLevel(                    /* low-level add entry                 */
HAUDFILE haud,                      /*   audit trail file to use           */
struct audEntry *src,               /*   formatted record to add           */
size_t srcSiz);                     /*   size of source buffer             */
```

Adds an already-formatted entry to an audit trail file.

haud
The file to which to add the record. This handle must have been gotten from a successful call to audfOpen(). The file must have been opened with the mode AUDMODE_READWRITE.

src
The record to add to the file. It must already have all fields filled in with valid quantities.

srcSiz
The size of the record to add. It should always be sizeof(struct audEntry).

Returns:
A code describing the result of the operation. It may be one of the following:

AUDERR_OK	The operation completed successfully.
AUDERR_NEW	The file was not found, so a new file was created. (This code will not be returned under normal conditions.)
AUDERR_DAMAGED	The file is damaged; you must recover the file before continuing. (This code will not be returned under normal conditions.)
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_MODE	The requested operation was not valid for the mode used when opening the file (e.g., attempting to write to a read-only file).

8.4.3. audAddfChannelStr ()

```
CHAR *                              /*   returns pointer to buffer        */
audfChannelStr(                     /* generate channel description string */
CHAR *buf,                          /*   buffer (must be AUDCHANSIZ long)  */
size_t bufSiz);                     /*   size of buffer                    */
                                    /*   (implicit inputs: usrnum, nterms, */
                                    /*    channel[], errcod)               */
```

Create a channel description string based on the current online context.

buf
The buffer to receive the channel description string. It should be at least AUDCHANSIZ (17) in size.

bufSiz
The actual size of the buf argument.

Returns:
A pointer to the buf argument.

8.4.4. audfclose()

```
VOID
audfClose(                          /* close an audit trail file           */
HAUDFILE haud);                     /*   handle to close                   */
```

Closes an audit trail file. The handle is no longer valid once this function has been called.

haud
The handle of the file to close. This handle must have been gotten from a successful call to audfOpen().

8.4.5. audfCvtEntryToOld()

```
struct audOldStyle *                /*   returns pointer to buffer         */
audfCvtEntryToOld(                  /* convert an entry to old-style       */
const struct audEntry *src,         /*   new-style audit trail entry       */
struct audOldStyle *buf,            /*   buffer to receive converted entry */
size_t bufSiz);                     /*   size of destination buffer        */
```

Converts a new (Worldgroup 3.12) audit trail entry to the old (Worldgroup 3.0 and earlier) format.

src
The buffer containing the new-style audit trail record.

buf
The buffer to receive the old-style audit trail record. Since the old Audit Trail typically used a simple character buffer to hold records, you can use a character buffer and typecast it when passing it to this function.

bufSiz
The actual size of the buffer to receive the old-style audit trail record. This should be at least AUDSIZ (132).

Returns:
A pointer to the buf argument.

8.4.6. audfCvtRawToEntry ()

```
struct audEntry *                   /*   returns pointer to buffer         */
audfCvtRawToEntry(                  /* convert raw file data to entry      */
const CHAR *src,                    /*   raw file data                     */
struct audEntry *buf,               /*   buffer to receive converted entry */
size_t bufSiz);                     /*   size of destination buffer        */
```

Converts raw audit trail file data to an audit trail entry structure.

src
The raw data, directly from an audit trail data file. This should be the entire line, including

space padding, and the carriage return and line feed at the end. (See above for details on the file format.)

buf
The buffer to receive the converted entry.

bufSiz
The actual size of the buf argument. It should always be sizeof(struct audEntry).

Returns:
A pointer to the buf argument.

8.4.7. audfFindEntryDOS()

```
INT                                 /*   returns result code               */
audfFindEntryDOS(                   /* find an entry by DOS date and time   */
HAUDFILE haud,                      /*   audit trail file to use            */
USHORT dosDate,                     /*   DOS-style date to search for       */
USHORT dosTime,                     /*   DOS-style time to search for       */
ULONG *pRecNum,                     /*   buf for rec # (NULL if don't want) */
struct audEntry *buf,               /*   buf for entry (NULL if don't want) */
size_t bufSiz);                     /*   size of destination buffer         */
```

Finds a specific audit trail record given a time and date stamp in DOS packed format. If an exact match is not found, the next highest record will be returned. If the target time is greater than the last record in the file, the last record will be returned. If there is more than one record with the same time stamp, the first one will be returned.

haud
The handle of the file to read from. This handle must have been gotten from a successful call to audfOpen().

dosDate
The date portion of the time stamp to search for, in DOS packed format.

dosTime
The time portion of the time stamp to search for, in DOS packed format.

pRecNum
Pointer to the buffer to receive the record number of the record when found. This argument should be NULL if you do not want to know the record number.

buf
The buffer to receive the record when found. This argument should be NULL if you do not want to read the actual record.

bufSiz
The actual size of the buf argument. It should always be sizeof(struct audEntry).

Returns:
A code describing the result of the operation. It may be one of the following:

AUDERR_OK	The operation completed successfully.
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it

	before you can use it.
AUDERR_NOTFOUND	The file is empty.

8.4.8. audfFindEntryStr()

```
INT                                 /*   returns result code                */
audfFindEntryStr(                   /* find an entry by string date & time  */
HAUDFILE haud,                      /*   audit trail file to use            */
const CHAR *strDate,                /*   YYYYMMDD date to search for        */
const CHAR *strTime,                /*   HHMMSS time to search for          */
ULONG *pRecNum,                     /*   buf for rec # (NULL if don't want) */
struct audEntry *buf,               /*   buf for entry (NULL if don't want) */
size_t bufSiz);                     /*   size of destination buffer         */
```

Finds a specific audit trail record given a time and date stamp in DOS packed format. If an exact match is not found, the next highest record will be returned. If the target time is greater than the last record in the file, the last record will be returned. If there is more than one record with the same time stamp, the first one will be returned.

haud
A handle to the Audit Trail file. This handle must have been gotten from a successful call to audfOpen().

strDate
The date portion of the time stamp to search for, in sort format.

strTime
The time portion of the time stamp to search for, in sort format.

pRecNum
Pointer to the buffer to receive the record number of the record when found. This argument should be NULL if you do not want to know the record number.

buf
The buffer to receive the record when found. This argument should be NULL if you do not want to read the actual record.

bufSiz
The actual size of the buf argument. It should always be sizeof(struct audEntry).

Returns:
A code describing the result of the operation. It may be one of the following:

AUDERR_OK	The operation completed successfully.
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_NOTFOUND	The file is empty.

8.4.9. audfFormatEntry ()

```
struct audEntry *                   /*   returns pointer to destination     */
audfFormatEntry(                    /* create properly-formatted entry      */
struct audEntry *buf,               /*   buffer to receive entry            */
size_t bufSiz,                      /*   size of destination buffer         */
const CHAR *brief,                  /*   brief description                  */
const CHAR *chanStr,                /*   channel description string         */
const CHAR *detail,                 /*   detailed description/format string */
va_list argList);                   /*   argument list                      */
```

Creates a properly-formatted audit trail record given its component parts.

buf
The buffer to receive the formatted record.

bufSiz
The actual size of the buf argument. It should always be sizeof(struct audEntry).

brief
The string to be used for the brief description. This is the same for each event of the same type. It will be truncated at AUDBRIEFSIZ-1 (32) characters.

chanStr
The string to be used for the channel description.

detail
A printf()-style format string which will be used to create the detailed description of the event. This field can be used to record event-specific information. It will be truncated at AUDDETSIZ-1 (127) characters. If it is truncated, the last character before the terminating nul character will be an asterisk ('*').

argList
A variable argument list created using the va_start() macro. This list along with the detail argument is used to create the detail field of the formatted record.

Returns:
A pointer to the buf argument.

8.4.10. audfNumRecs ()

```
ULONG
audfNumRecs(                        /* get number of records in file        */
HAUDFILE haud);                     /*   audit trail file to use            */
```

Gets the number of records in an audit trail file.

haud
A handle to the Audit Trail file. This handle must have been gotten from a successful call to audfOpen().

Returns:
The number of records in the file.

8.4.11. audfOpen()

```
INT                                 /*   returns result code                */
audfOpen(                           /* open an audit trail file             */
HAUDFILE *phaud,                    /*   buffer to receive opened handle    */
const CHAR *fileName,               /*   name of file to open               */
AUDMODE mode);                      /*   mode in which to open              */
```

Opens an audit trail file. The file may be opened in one of the following modes:

AUDMODE_READONLY	Open with read-only access to the file.
AUDMODE_READWRITE	Open with read and write access to the file.

If the file does not exist and you specify read-only mode, the function will return an I/O error response. If the file does not exist and you specify read/write mode, the file will be created and the function will return a file-created response.

If the file has been damaged, it will still be opened, but the function will return a file-damaged response. In this case, you must recover the file before you will be able to read from it.

phaud
Pointer to the buffer to receive a handle to the opened file.

fileName
The complete path and file name of the file to open.

mode
The mode in which to open the file. This must be either AUDMODE_READONLY or AUDMODE_READWRITE.

Returns:

AUDERR_OK	The operation completed successfully.
AUDERR_NEW	The specified file was not found, so a new file was created.
AUDERR_DAMAGED	The file is damaged. You must recover the file before proceeding.
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error. This response will be returned if you try to open a non-existent file in read-only mode.
AUDERR_MEMORY	There was not enough memory to open the file.

8.4.12. audfReadEntry()

```
INT                                 /*   returns result code                */
audfReadEntry(                      /* read an entry by record number       */
HAUDFILE haud,                      /*   audit trail file to use            */
```

```
ULONG recNum,                       /*   record number to read             */
struct audEntry *buf,               /*   buffer to receive data            */
size_t bufSiz);                     /*   size of destination buffer        */
```

Reads a record from an audit trail file.

haud
A handle of the Audit Trail file. This handle must have been gotten from a successful call to audfOpen().

recNum
The record number to read. This must be between zero and one less than the total number of records in the file.

buf
The buffer to receive the record.

bufSiz
The actual size of the buf argument. It should always be sizeof(struct audEntry).

Returns:

AUDERR_OK	The record was read successfully.
AUDERR_FIO	An I/O error occurred while reading the record. Check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_NOTFOUND	The requested record was not found (the record number was out of range).

8.4.13. audfReadLowLevel()

```
INT                                 /*   returns result code               */
audfReadLowLevel(                   /* low-level read an entry utility     */
HAUDFILE haud,                      /*   audit trail file to use           */
ULONG recNum,                       /*   record number to read             */
CHAR *buf,                          /*   buffer (must be AUDRECLEN+1 long) */
size_t bufSiz);                     /*   size of buffer                    */
```

Reads a raw data record from an audit trail file but does not convert it into the standard entry structure. The raw data can be converted into an entry structure using audfCvtRawToEntry().

haud
A handle to the Audit Trail file. This handle must have been gotten from a successful call to audfOpen().

recNum
The record number to read. This must be between zero and one less than the total number of records in the file.

buf
The buffer to receive the raw data, directly from the audit trail data file. This will be the entire line, including space padding, and the carriage return and line feed at the end. (See above for details on the file format.) This buffer should be at least AUDRECLEN+1 (195) in size.

bufSiz
The actual size of the buf argument.

Returns:

AUDERR_OK	The record was read successfully.
AUDERR_FIO	An I/O error occurred while reading the record. Check the global variable errno for the specific error.

AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_NOTFOUND	The requested record was not found (the record number was out of range).

8.4.14. audfReadOldStyle()

```
INT                                 /*   returns result code               */
audfReadOldStyle(                   /* read an old-style entry             */
HAUDFILE haud,                      /*   audit trail file to use           */
ULONG recNum,                       /*   record number to read             */
struct audOldStyle *buf,            /*   buffer to receive data            */
size_t bufSiz);                     /*   size of destination buffer        */
```

Reads a record from an audit trail file and converts to the old style (pre-Worldgroup 3.1).

haud
A handle to the Audit Trail file. This handle must have been gotten from a successful call to audfOpen().

recNum
The record number to read. This must be between zero and one less than the total number of records in the file.

buf
The buffer to receive the old-style audit trail record. Since the old Audit Trail typically used a simple character buffer to hold records, you can use a character buffer and typecast it when passing it to this function.

bufSiz
The actual size of the buffer to receive the old-style audit trail record. This should be at least AUDSIZ (132).

Returns:

AUDERR_OK	The record was read successfully.
AUDERR_FIO	An I/O error occurred while reading the record. Check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_NOTFOUND	The requested record was not found (the record number was out of range).

8.4.15. audfRecover()

```
INT                                 /*   returns result code               */
audfRecover(                        /* recover a damaged audit trail file  */
HAUDFILE haud);                     /*   handle to file to recover         */
```

Recovers a damaged audit trail file.

haud
The handle of the file to recover. This handle must have been gotten from a call to audfOpen() that returned AUDERR_DAMAGED.

Returns:

AUDERR_OK	The operation completed successfully.
AUDERR_NEW	The specified file was not found, so a new file was created. (This code will not be returned under normal conditions.)
AUDERR_DAMAGED	The file is still damaged. (This code will not be returned under normal conditions.)
AUDERR_FIO	An I/O error occurred while recovering the file; check the global variable errno for the specific error.

8.4.16. audfVAddEntry()

```
INT                                 /*   returns result code                */
audfVAddEntry(                      /* add entry with variable argument list*/
HAUDFILE haud,                      /*   audit trail file to use            */
const CHAR *brief,                  /*   brief description                  */
const CHAR *detail,                 /*   detailed description/format string */
va_list argList);                   /*   argument list                      */
```

Adds a record to an audit trail file; does not take variable arguments. It requires an online context as discussed above.

haud
The file to which to add the record. This handle must have been gotten from a successful call to audfOpen(). The file must have been opened with the mode AUDMODE_READWRITE.

brief
A brief description of the event. This is the same for each event of the same type. It will be truncated at AUDBRIEFSIZ-1 (32) characters.

detail
Details on the event. This is a printf()-style format string which can be used to record event-specific information. It will be truncated at AUDDETSIZ-1 (127) characters. If it is truncated, the last character before the terminating nul character will be an asterisk ('*').

argList
A variable argument list created using the va_start() macro. This list along with the detail argument is used to create the detail field of the formatted record.

Returns:
A code describing the result of the operation. It may be one of the following:

AUDERR_OK	The operation completed successfully.
AUDERR_NEW	The file was not found, so a new file was created. (This code will not be returned under normal conditions.)
AUDERR_DAMAGED	The file is damaged; you must recover the file before continuing. (This code will not be returned under normal conditions.)
AUDERR_FIO	An I/O error occurred while performing an operation; check the global variable errno for the specific error.
AUDERR_NOTOPEN	The file is not open, you must close and reopen it before you can use it.
AUDERR_MODE	The requested operation was not valid for the mode used when opening the file (e.g., attempting to write to a read-only file).

9. Date and Time API

9.1. Overview

This document outlines changes and additions to the Date and Time API made for Worldgroup 3.12. This document also covers changes to the VB Date and Time API.

The most important changes were For Year 2000 compliance, and the addition of the discrete components format and sort format for dates and times. Many of the functions which take pointers as arguments or return pointers to internal buffers now specify that these are const pointers. A complete list of functions to which const was added will not be given in this document.

9.1.1. Year 2000 Compliance

The primary changes made for year 2000 compliance are improved support for four-digit years in string-format dates. The following defines and functions were changed or added for year 2000 compliance:

Name	Type	Status
dcdate	Function	Changed
LDATSIZ	Define	Added
ncdatel	Function	Added
ncedatl	Function	Added
v2sdatl	Function	Added

9.1.2. Discrete Components Date and Time Format

The most basic binary representation of a date or time is to use separate integer variables to hold the year, month, and day components of a date, and the hour, minute, and second components of a time. This format has now been formalized and will be referred to as the "discrete components" format.

The components of a discrete date are defined as follows:

Component	**Range**	**Comments**
Year	1582-9999	This is the calendar year. Since the Gregorian calendar was first adopted on October 15, 1582, earlier dates should not be manipulated with this API.
Month	1-12	The one-based month, where one is January and twelve is December.
Day	1-31	The one-based day of the month, where one is the first day of the month. The actual maximum value of this component depends on the month and year.

The components of a discrete time are defined as follows:

Component	**Range**	**Comments**
Hour	0-23	The zero-based hour of the day in 24-hour time (military time), where zero is 12:00 AM (midnight), 12 is 12:00 PM (noon), and 23 is 11:00 PM.
Minute	0-59	The zero-based minute of the hour, where the first minute in an hour is zero.
Second	0-59	The zero-based second of the minute, where the first second in a minute is zero.

The following identifiers have been changed or added to support the discrete components format:

Name	Type	Status

addDaysToDate	Function	Added
dateDecode	Function	Added
dayFromDate	Function	Added
difDate	Function	Added
difYear	Function	Added
lastDayOfMonth	Function	Added
prnDate	Function	Added
prnTime	Function	Added
timeDecode	Function	Added
validDate	Function	Added
validTime	Function	Added

9.1.3. Sort Date and Time String Format

In order to provide a date and time string format that is compact, provides a complete representation of the date, and can be used to sort dates easily, the sort date and time string format was defined.

The sort date format consists of a four-digit year, immediately followed by a two-digit month, immediately followed by a two-digit day. For example, January 2, 1980 is represented by "19800102".

The sort time format consists of a two-digit hour (in 24-hour format), immediately followed by a two-digit minute, immediately followed by a two-digit second. For example, three seconds before midnight (11:59:57 PM) is represented by "235957".

The following identifiers have been changed or added to support the sort format:

Name	Type	Status
sDateDecode	Function	Added
sDateDecodeDOS	Function	Added
sDateEncode	Function	Added
sDateEncodeDOS	Function	Added
sDecodeDT	Function	Added
sDecodeDTDOS	Function	Added
sTimeDecode	Function	Added
sTimeDecodeDOS	Function	Added
sTimeEncode	Function	Added
sTimeEncodeDOS	Function	Added

9.1.4. Miscellaneous

A number of identifiers have been changed or added for other reasons. These are as follows:

Name	Type	Status
DATSIZ	Define	Added
isLeapYear	Define	Added
PRND_*	Define	Added
PRNT_*	Define	Added
strDays	Variable	Added
strMonths	Variable	Added
validDateDOS	Function	Added
validTimeDOS	Function	Added

9.2. Defines

9.2.1. DATSIZ

```
#define DATSIZ 9                    /* size of a date in MM/DD/YY format    */
```

This define was originally declared in USRACC.H. It has been moved to DNTAPI.H.

9.2.2. isLeapYear

```
#define isLeapYear(yr)              /* is yr a leap year?                   */
```

This is a function-style macro that examines a year and determines whether it is a leap year based on the rules defined for the Gregorian calendar.

yr
The year to examine. It can be any integer type.

Returns:
A non-zero integer value if the year is a leap year, or zero if it is not a leap year.

9.2.3. LDATSIZ

```
#define LDATSIZ 11                  /* size of a date in MM/DD/YYYY format  */
```

The size of a character buffer required to hold a U.S. standard format date with a four-digit year (MM/DD/YYYY).

9.2.4. PRND_*, PRNT_*

```
#define PRND_*                      /* prnDate format identifiers           */
#define PRNT_*                      /* prnTime format identifiers           */
```

These defines have been added to make it easier to use the prndat(), prntim(), prnDate(), and prnTime() functions. See the prnDate() and prnTime() functions for details.

9.3. Variables

9.3.1. strMonths

```
CHAR strMonths[12][10];             /* long month names (0 = January)       */
```

A zero-based array of month names in English. The array is defined as follows:

Index	Value
0	"January"
1	"February"
2	"March"
3	"April"
4	"May"
5	"June"
6	"July"
7	"August"
8	"September"
9	"October"
10	"November"
11	"December"

9.3.2. strDays

```
CHAR strDays[7][10];                /* long day names (0 = Sunday)          */
```

A zero-based array of names of the days of the week in English. The array is defined as follows:

Index	Value
0	"Sunday"
1	"Monday"
2	"Tuesday"
3	"Wednesday"
4	"Thursday"
5	"Friday"
6	"Saturday"

9.4. Functions

9.4.1. addDaysToDate

```
VOID
addDaysToDate(                      /* get date by adding # days to base    */
LONG nDays,                         /*   # days to add (can be negative)    */
INT *pYear,                         /*   base year (updated w/ new year)    */
INT *pMonth,                        /*   base month (updated w/ new month)  */
INT *pDay);                         /*   base day (updated w/ new day)      */
```

Adds days to or subtracts days from a date in discrete components. If the starting date is invalid or the number of days is out of range (approximately -152000 to 2922365 as of January 1998), the results are undefined.

nDays
The number of days to add to the date. If this is negative, the days will be subtracted from the date.

pYear
Pointer to a buffer containing the year component of the date. This buffer is updated with the new year.

pMonth
Pointer to a buffer containing the month component of the date. This buffer is updated with the new month.

pDay
Pointer to a buffer containing the day component of the date. This buffer is updated with the new day.

9.4.2. dateDecode

```
VOID
dateDecode(                         /* decode date from "MM/DD[/YY[YY]]"    */
const CHAR *src,                    /*   date string                        */
INT *pYear,                         /*   buffer for year (-1 if bad)        */
INT *pMonth,                        /*   buffer for month (-1 if bad)       */
INT *pDay);                         /*   buffer for day (-1 if bad)         */
```

Decodes a U.S. standard date string (MM/DD/YYYY) into discrete components. The year can be two digits, four digits, or not specified. If two digits are used, values less than 80 will be assumed to be in the 2000s and values greater-than or equal-to 80 will be assumed to be in the 1900s. If no year is specified, the current year will be assumed.

src

The U.S. standard date string to decode.

pYear
Pointer to a buffer to receive the year component of the date. If the year component of the date string is invalid, this buffer will be filled with -1.

pMonth
Pointer to a buffer to receive the month component of the date. If the month component of the date string is invalid, this buffer will be filled with -1.

pDay
Pointer to a buffer to receive the day component of the date. If the day component of the date string is invalid, this buffer will be filled with -1.

9.4.3. dayFromDate

```
INT                                 /*   day of week (0 = Sun, <0 = error)  */
dayFromDate(                        /* get day of week given date           */
INT year,                           /*   year component (1-9999)            */
INT month,                          /*   month component (1-12)             */
INT day);                           /*   day component (1-31)               */
```

Determines the day of the week represented by a date specified in discrete components.

year
The year component of the date.

month
The month component of the date.

day
The day component of the date.

Returns:
A zero-based index describing the day of the week (Sunday = 0, Saturday = 6). If the specified date is not valid, a negative value will be returned.

9.4.4. dcdate

```
USHORT                              /*   returns GCINVALIDDOT if invalid    */
dcdate(                             /* decode DOS date from "MM/DD[/YY[YY]]"*/
const CHAR *datstr);                /*   date string to convert             */
```

Decodes a U.S. standard date string (MM/DD/YYYY) into a DOS packed date. The year can be two digits, four digits, or not specified. If two digits are used, values less than 80 will be assumed to be in the 2000s and values greater-than or equal-to 80 will be assumed to be in the 1900s. If no year is specified, the current year will be assumed.

Note: this function was changed between Worldgroup 3.0 and 3.12 to accept four-digit years in the input string.

datstr
The U.S. standard date string to decode.

Returns:
The date in DOS packed format. If the input date string is not valid, it returns GCINVALIDDOT (0xFFFF).

9.4.5. difDate

```
LONG                                /*   < 0 if end date before beg date    */
difDate(                            /* compute # days between dates         */
INT begYear,                        /*   year component of first date       */
INT begMonth,                       /*   month component of first date      */
INT begDay,                         /*   day component of first date        */
INT endYear,                        /*   year component of second date      */
INT endMonth,                       /*   month component of second date     */
INT endDay);                        /*   day component of second date       */
```

Computes the number of days between two dates specified in discrete components. If the ending date comes before the beginning date, the result is negative. If either of the dates are not valid, the result is undefined.

begYear
The year component of the beginning date.

begMonth
The month component of the beginning date.

begDay
The day component of the beginning date.

endYear
The year component of the ending date.

endMonth
The month component of the ending date.

endDay
The day component of the ending date.

Returns:
The number of days between the beginning date and the ending date.

9.4.6. difYear

```
LONG                                /*   < 0 if end year before beg year    */
difYear(                            /* get # days between 1JAN of two years */
INT begYear,                        /*   beginning year                     */
INT endYear);                       /*   ending year                        */
```

Computes the number of days between January 1st of two years. If the ending year comes before the beginning year, the result is negative. If either year is out of range (1582 to 9999), the result is undefined.

begYear
The beginning year.

endYear
The ending year.

Returns:
The number of days between January 1st of the beginning and ending years.

9.4.7. lastDayOfMonth

```
INT                                 /*   number of last day (e.g. Jan = 31) */
lastDayOfMonth(                     /* get highest day in given month/year  */
```

```
INT month,                          /*   number of month (1-12)            */
INT year);                          /*   year in question (1-9999)         */
```

Determine the last day of a month. For example, January is always 31 and September is always 30, while February may be 28 or 29, depending on whether or not it is a leap year.

month
The month for which the last day is desired.

year
The year in which the month occurs.

Returns:
The number of the last day of the month (which is also the number of days in the month).

9.4.8. ncdatel

```
const CHAR *                        /*   returns ASCII long date string    */
ncdatel(                            /* encodes date into "MM/DD/YYYY"      */
USHORT date);                       /*   packed date                       */
```

Encodes a DOS packed date into a U.S. standard (MM/DD/YYYY) date string with a four-digit year. If the date specified is not valid, an empty string is returned.

date
The DOS packed format date to encode.

Returns:
A pointer to a single internal buffer containing the date string.

9.4.9. ncedatl

```
const CHAR *                        /*   date string in European long format*/
ncedatl(                            /* encodes date to DD-MMM-YYYY         */
USHORT date);                       /*   packed date                       */
```

Encodes a DOS packed date into a European (DD-MMM-YYYY) date string with a four-digit year. If the date specified is not valid, an empty string is returned.

date
The DOS packed format date to encode.

Returns:
A pointer to a single internal buffer containing the date string.

9.4.10. prnDate

```
CHAR *                              /*   returns pointer to buffer         */
prnDate(                            /* form formatted date string         */
INT year,                           /*   year component (1-9999)           */
INT month,                          /*   month component (1-12)            */
INT day,                            /*   day component (1-31)              */
CHAR *buf,                          /*   buffer to receive formatted date  */
size_t bufSiz,                      /*   size of buffer                    */
INT mode,                           /*   format mode                       */
CHAR sep);                          /*   separator character               */
```

Encodes a date specified in discrete components into a string in a variety of formats. If the date specified is

not valid, an empty string is generated.

year
The year component of the date to encode.

month
The month component of the date to encode.

day
The day component of the date to encode.

buf
A pointer to the buffer to receive the encoded date string.

bufSiz
The actual size of the buf argument.

mode
A code describing the format to use when encoding the date. A standard set of modes are defined; all such identifiers begin with PRND_. The rest of the identifier is constructed as follows:

Identifier	Result
Y	A two-digit numeric representation of the year. This representation is always zero-padded if necessary (e.g., 2001 is represented by "01").
YY	A four-digit numeric representation of the year.
M	A numeric representation of the month. This representation is always zero-padded if necessary (e.g., January is represented by "01").
MM	A short, three-character string representation of the month name (e.g., January is represented by "Jan").
MMM	A full string representation of the month name (e.g., January is represented by "January").
C	Following a string-form month specifier means that the month name should be capitalized (e.g., MMC means the date string should encode January as "JAN").
D	A numeric representation of the day. This representation is always zero-padded if necessary (e.g., the first is represented by "01").

All possible combinations of these identifiers are not supported. The following combinations are currently supported:

Identifier	Value	Date Format
PRND_MD	0	12*31
PRND_MY	2	12*90
PRND_MDY	4	12*31*90
PRND_DMMY	6	31*Dec*90
PRND_DMMCY	7	31*DEC*90
PRND_MMDY	8	Dec 31, 90
PRND_MMCDY	9	DEC 31, 90
PRND_MMMDY	10	December 31, 90
PRND_MMMCDY	11	DECEMBER 31, 90
PRND_MYY	14	12*1990
PRND_MDYY	16	12*31*1990
PRND_DMMYY	18	31*Dec*1990
PRND_DMMCYY	19	31*DEC*1990
PRND_MMDYY	20	Dec 31, 1990
PRND_MMCDYY	21	DEC 31, 1990
PRND_MMMDYY	22	December 31, 1990
PRND_MMMCDYY	23	DECEMBER 31, 1990

sep
The character to use as a separator for certain formats. This character replaces the asterisk (*) in the above table of supported formats.

Returns:
A pointer to the buf argument.

9.4.11. prnTime

```
CHAR *                          /*   returns pointer to buffer          */
prnTime(                        /* form formatted time string           */
INT hour,                       /*   hour component (0-23)              */
INT minute,                     /*   minute component (0-59)            */
INT second,                     /*   second component (0-59)            */
CHAR *buf,                      /*   buffer to receive formatted time   */
size_t bufSiz,                  /*   size of buffer                     */
INT mode);                      /*   format mode                        */
```

Encodes a time specified in discrete components into a string in a variety of formats. If the time specified is not valid, an empty string is generated.

hour
The hour component of the time to encode.

minute
The minute component of the time to encode.

second
The second component of the time to encode.

buf
A pointer to the buffer to receive the encoded time string.

bufSiz
The actual size of the buf argument.

mode
A code describing the format to use when encoding the time. A standard set of modes are defined; all such identifiers begin with PRNT_. The rest of the identifier is constructed as follows:

Identifier	Result
H	A numeric representation of the hour. This may be in 12- or 24-hour format, and it is not zero-padded by default.
M	A numeric representation of the minute. This representation is always zero-padded if necessary (e.g., the third minute of an hour, minute two, is represented by "02").
S	A numeric representation of the second. This representation is always zero-padded if necessary (e.g., the third second of a minute, second two, is represented by "02").
_	A single space character.
P	The AM/PM portion of the time represented by a single character. If this is specified, the hour is given in 12-hour format.
PM	The AM/PM portion of the time represented by two characters. If this is specified, the hour is given in 12-hour format.
C	Following P or PM specifies that the AM/PM will be capitalized.

All possible combinations of these identifiers are not supported. The following combinations are currently supported:

Identifier	Value	Time Format
PRNT_HM	0	23:59
PRNT_HMP	1	11:59p
PRNT_HMPM	2	11:59pm
PRNT_HM_P	3	11:59 p
PRNT_HM_PM	4	11:59 pm
PRNT_HMPC	6	11:59P
PRNT_HMPMC	7	11:59PM
PRNT_HM_PC	8	11:59 P
PRNT_HM_PMC	9	11:59 PM
PRNT_HMS	10	23:59:59
PRNT_HMSP	11	11:59:59p
PRNT_HMSPM	12	11:59:59pm
PRNT_HMS_P	13	11:59:59 p
PRNT_HMS_PM	14	11:59:59 pm
PRNT_HMSPC	16	11:59:59P
PRNT_HMSPMC	17	11:59:59PM
PRNT_HMS_PC	18	11:59:59 P
PRNT_HMS_PMC	19	11:59:59 PM
PRNT_PAD	20	Add this to one of the above identifiers to zero-pad hour

Returns:
A pointer to the buf argument.

9.4.12. sDateDecode

```
VOID
sDateDecode(                        /* decode "YYYYMMDD"                    */
const CHAR *src,                    /*   encoded date string                */
INT *pYear,                         /*   buffer for year (-1 if bad)        */
INT *pMonth,                        /*   buffer for month (-1 if bad)       */
INT *pDay);                         /*   buffer for day (-1 if bad)         */
```

Decodes a sort format date string into discrete components.

src
The sort format date string to decode.

pYear
Pointer to a buffer to receive the year component of the date. If the year component of the date string is invalid, this buffer will be filled with -1.

pMonth
Pointer to a buffer to receive the month component of the date. If the month component of the date string is invalid, this buffer will be filled with -1.

pDay
Pointer to a buffer to receive the day component of the date. If the day component of the date string is invalid, this buffer will be filled with -1.

9.4.13. sDateDecodeDOS

```
USHORT                              /*   DOS date or GCINVALIDDOT           */
sDateDecodeDOS(                     /* decode "YYYYMMDD" to DOS format      */
const CHAR *src);                   /*   encoded date string                */
```

Decodes a sort format date string into a DOS packed date.

src
The sort format date string to decode.

Returns:
The date in DOS packed format. If the input date string is not valid, it returns GCINVALIDDOT (0xFFFF).

9.4.14. sDateEncode

```
CHAR *                          /*   returns pointer to buffer          */
sDateEncode(                    /* encode date to "YYYYMMDD"            */
INT year,                       /*   year component (1-9999)            */
INT month,                      /*   month component (1-12)             */
INT day,                        /*   day component (1-31)               */
CHAR *buf,                      /*   buffer to receive encoded date     */
size_t bufSiz);                 /*   size of buffer                     */
```

Encodes a date in discrete components to a sort format date string.

year
The year component of the date to encode.

month
The month component of the date to encode.

day
The day component of the date to encode.

buf
A pointer to the buffer to receive the encoded date string.

bufSiz
The actual size of the buf argument.

Returns:
A pointer to the buf argument.

9.4.15. sDateEncodeDOS

```
CHAR *                          /*   returns pointer to buffer          */
sDateEncodeDOS(                 /* encode DOS date to "YYYYMMDD"        */
USHORT dosDate,                 /*   DOS packed date                    */
CHAR *buf,                      /*   buffer to receive encoded date     */
size_t bufSiz);                 /*   size of buffer                     */
```

Encodes a date in DOS packed format to a sort format date string.

dosDate
The date to encode in DOS packed format.

buf
A pointer to the buffer to receive the encoded date string.

bufSiz
The actual size of the buf argument.

Returns:

A pointer to the buf argument.

9.4.16. sDecodeDT

```
VOID
sDecodeDT(                          /* decode "YYYYMMDD[ ]HHMMSS"           */
const CHAR *src,                    /*   encoded date string                */
INT *pYear,                         /*   buffer for year (-1 if bad)        */
INT *pMonth,                        /*   buffer for month (-1 if bad)       */
INT *pDay,                          /*   buffer for day (-1 if bad)         */
INT *pHour,                         /*   buffer for hour (-1 if bad)        */
INT *pMinute,                       /*   buffer for minute (-1 if bad)      */
INT *pSecond);                      /*   buffer for second (-1 if bad)      */
```

Decodes a string containing a sort format date followed by a sort format time into a date and time in discrete components.

src
The string to decode. It must consist of a sort format date, followed by zero or more whitespace characters, followed by a sort format time.

pYear
Pointer to a buffer to receive the year component of the date. If the year component of the date string is invalid, this buffer will be filled with -1.

pMonth
Pointer to a buffer to receive the month component of the date. If the month component of the date string is invalid, this buffer will be filled with -1.

pDay
Pointer to a buffer to receive the day component of the date. If the day component of the date string is invalid, this buffer will be filled with -1.

pHour
Pointer to a buffer to receive the hour component of the time. If the hour component of the time string is invalid, this buffer will be filled with -1.

pMinute
Pointer to a buffer to receive the minute component of the time. If the minute component of the time string is invalid, this buffer will be filled with -1.

pSecond
Pointer to a buffer to receive the second component of the time. If the second component of the time string is invalid, this buffer will be filled with -1.

9.4.17. sDecodeDTDOS

```
VOID
sDecodeDTDOS(                       /* decode "YYYYMMDD[ ]HHMMSS" to DOS fmt*/
const CHAR *src,                    /*   encoded date string                */
USHORT *pDate,                      /*   buf for date (GCINVALIDDOT if bad) */
USHORT *pTime);                     /*   buf for time (GCINVALIDDOT if bad) */
```

Decodes a string containing a sort format date followed by a sort format time into a date and time in DOS packed format.

src
The string to decode. It must consist of a sort format date, followed by zero or more whitespace characters,

followed by a sort format time.

pDate
Pointer to a buffer to receive the date in DOS packed format. If the date is not valid, this buffer will be filled with GCINVALIDDOT (0xFFFF).

pTime
Pointer to a buffer to receive the time in DOS packed format. If the time is not valid, this buffer will be filled with GCINVALIDDOT (0xFFFF).

9.4.18. sTimeDecode

```
VOID
sTimeDecode(                    /* decode "HHMMSS"                  */
const CHAR *src,                /*   encoded date string            */
INT *pHour,                     /*   buffer for hour (-1 if bad)    */
INT *pMinute,                   /*   buffer for minute (-1 if bad)  */
INT *pSecond);                  /*   buffer for second (-1 if bad)  */
```

Decodes a sort format time string into discrete components.

src
The sort format time string to decode.

pHour
Pointer to a buffer to receive the hour component of the time. If the hour component of the time string is invalid, this buffer will be filled with -1.

pMinute
Pointer to a buffer to receive the minute component of the time. If the minute component of the time string is invalid, this buffer will be filled with -1.

pSecond
Pointer to a buffer to receive the second component of the time. If the second component of the time string is invalid, this buffer will be filled with -1.

9.4.19. sTimeDecodeDOS

```
USHORT                          /*   DOS time or GCINVALIDDOT       */
sTimeDecodeDOS(                 /* decode "HHMMSS" to DOS format    */
const CHAR *src);               /*   encoded time string            */
```

Decodes a sort format time string into a DOS packed time.

src
The sort format time string to decode.

Returns:
The date in DOS packed format. If the input date string is not valid, it returns GCINVALIDDOT (0xFFFF).

9.4.20. sTimeEncode

```
CHAR *                          /*   returns pointer to buffer      */
sTimeEncode(                    /* encode time to "HHMMSS"          */
INT hour,                       /*   hour component (0-23)          */
INT minute,                     /*   minute component (0-59)        */
INT second,                     /*   second component (0-59)        */
CHAR *buf,                      /*   buffer to receive encoded time */
```

```
size_t bufSiz);                     /*   size of buffer                     */
```

Encodes a time in discrete components to a sort format time string.

hour
The hour component of the time to encode.

minute
The minute component of the time to encode.

second
The second component of the time to encode.

buf
A pointer to the buffer to receive the encoded time string.

bufSiz
The actual size of the buf argument.

Returns:
A pointer to the buf argument.

9.4.21. sTimeEncodeDOS

```
CHAR *                              /*   returns pointer to buffer          */
sTimeEncodeDOS(                     /* encode DOS time to "HHMMSS"          */
USHORT dosTime,                     /*   DOS packed time                    */
CHAR *buf,                          /*   buffer to receive encoded time     */
size_t bufSiz);                     /*   size of buffer                     */
```

Encodes a time in DOS packed format to a sort format time string.

dosTime
The time to encode in DOS packed format.

buf
A pointer to the buffer to receive the encoded time string.

bufSiz
The actual size of the buf argument.

Returns:
A pointer to the buf argument.

9.4.22. timeDecode

```
VOID
timeDecode(                         /* decode time from  "HH:MM[:SS]"       */
const CHAR *src,                    /*   encoded date string                */
INT *pHour,                         /*   buffer for hour (-1 if bad)        */
INT *pMinute,                       /*   buffer for minute (-1 if bad)      */
INT *pSecond);                      /*   buffer for second (-1 if bad)      */
```

Decodes a standard date string (HH:MM:SS) into discrete components. The seconds portion of the time string is optional. If no second is specified, 00 will be assumed.

src
The standard time string to decode.

pHour
Pointer to a buffer to receive the hour component of the time. If the hour component of the time string is invalid, this buffer will be filled with -1.

pMinute
Pointer to a buffer to receive the minute component of the time. If the minute component of the time string is invalid, this buffer will be filled with -1.

pSecond
Pointer to a buffer to receive the second component of the time. If the second component of the time string is invalid, this buffer will be filled with -1.

9.4.23. v2sdatl

```
const CHAR *                        /*   returns ptr to MM/DD/YYYY          */
v2sdatl(                            /* convert VB date to stg "MM/DD/YYYY"  */
DOUBLE vbdat);                      /*   VB DOUBLE precision date/time      */
```

Encodes a VB binary date into a U.S. standard (MM/DD/YYYY) date string with a four-digit year. If the date specified is not valid, an empty string is returned.

vbdat
The VB binary format date to encode.

Returns:
A pointer to a single internal buffer containing the date string.

9.4.24. validDate

```
GBOOL
validDate(                          /* check for valid date                 */
INT year,                           /*   year component (1-9999)            */
INT month,                          /*   month component (1-12)             */
INT day);                           /*   day component (1-31)               */
```

Test a date in discrete components for validity.

year
The year component of the date to test. The year must be in the range 1 to 9999 for the date to be valid.

month
The month component of the date to test. The month must be in the range 1 to 12 for the date to be valid.

day
The day component of the date to test. The day must be greater-than or equal-to one, and it must be less-than or equal-to the last day of the month for the date to be valid. The last day of the month is determined from the month and year, taking leap years into account.

Returns:
A non-zero value if the date is valid, or zero if the date is not valid.

9.4.25. validDateDOS

```
GBOOL
validDateDOS(                       /* check for valid DOS date             */
USHORT dosDate);                    /*   DOS packed date                    */
```

Test a date in DOS packed format for validity.

dosDate
The date to test in DOS packed format.

Returns:
A non-zero value if the date is valid, or zero if the date is not valid.

9.4.26. validTime

```
GBOOL
validTime(                          /* check for valid time                */
INT hour,                           /*   hour component (0-23)             */
INT minute,                         /*   minute component (0-59)           */
INT second);                        /*   second component (0-59)           */
```

Test a time in discrete components for validity.

hour
The hour component of the time to test. The hour must be in the range 0 to 23 for the time to be valid.

minute
The minute component of the time to test. The minute must be in the range 0 to 59 for the time to be valid.

second
The second component of the time to test. The second must be in the range 0 to 59 for the time to be valid.

Returns:
A non-zero value if the date is valid, or zero if the date is not valid.

9.4.27. validTimeDOS

```
GBOOL
validTimeDOS(                       /* check for valid DOS time            */
USHORT dosTime);                    /*   DOS packed time                   */
```

Test a time in DOS packed format for validity.

dosTime
The time to test in DOS packed format.

Returns:
A non-zero value if the date