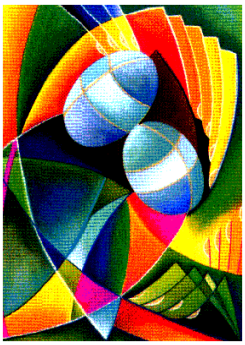
MattOpen 2.16



Developer's Guide

2005

Copy Restrictions

A This software and the accompanying written materials are copyrighted and are proprietary products of Skybeam. Copying of the software and of the written materials is prohibited. Subject to these restrictions, you may make one copy of the software for backup or archival purposes. That backup/archival copy must have a label placed on the magnetic media, showing the program name, and the copyright and trademark designation in the same form as the original software.

B You may not decompile, disassemble, reverse engineer, copy, transfer, or otherwise use the software except as expressly stated in this Agreement.

Disclaimer of Warranty

The software and accompanying written materials (including instructions for use) are provided "As Is" without warranty of any kind. Skybeam makes no warranties, express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. No oral or written information or advice given by Skybeam, its dealers, distributors, agents or employees shall create a warranty, and you may not rely on any such information or advice.

Limitation of Remedies

In no event shall Skybeam be liable to you for any damages, including any loss of profits, or other identical or consequential damages, arising out of your use of or inability to use the software or the written materials, even if Skybeam has been advised of the possibility of such damages.

Copyright

Hotline

Skybeam Management Ltd.

Tel: +1 877 731 0114
Fax:+44 808 208 3500

Distribution

Skybeam Management Ltd.

Tel: +1 877 731 0114
Fax: +44 808 208 3500
www.skybeam.biz

Table of Contents

Preface

These years, many Tandem customers face a number of critical and decisive choices related to IT. Should one maintain, develop, or further develop their system? Should one opt for safe and well-known technology or migrate, or should one combine and optimize the existing technology with the latest? MattOpen 2.0 is a series of development tools that may assist you in reaching the right conclusions when addressing these choices.

MattOpen 2.0 is one of the components of the Matterhorn Suite. Together, these components provide Tandem customers with Pathway applications with easy and effortless access to the client/server and Internet/intranet technologies – and, what is more, without the customer having to rewrite a single line of code in their existing Pathway applications.

MattOpen 2.0 is composed of three independent development tools; *MattOpen Dynamic Interface*, *Requester Replacer*, and the DDE- and DLL statements of the *Matterhorn Macro Language*. When used optimally, these tools, together with the rest of the Matterhorn Suite - will make your Tandem system as open to new technologies as it will ever get. Using the tools of MattOpen 2.0 you may:

- Access and control legacy Pathway applications on the Tandem from newer client applications on the PC.

- Redirect requester calls from legacy Pathway applications to procedures in a Windows DLL. In effect, you *replace* the entire requester with a DLL. The feature enables you to create and tailor new powerful applications without risking past investments.

- Place calls in legacy Pathway applications to procedures in a Windows DLL. The DLL is called during the execution of a macro and the feature extends the functionality of the current requester, enabling you for instance to retrieve values from a remote database or validate contents of fields.

- Create and run macros in order to set up DDE conversations between your legacy Pathway applications and any DDE-compatible application.

In short, the tools of MattOpen 2.0 provide gateways to and from legacy Pathway applications. Using MattOpen, Tandem customers may continue to create and add faster and more powerful client applications in programming languages like Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or Borland Pascal and integrate them with their existing Pathway applications. Your legacy Pathway system may be integrated with new PC-based applications with a minimum of investment and risk.

About This Manual

Before you start using the MattOpen development tools, please take your time to study this manual. The manual provides an overview of MattOpen features and operation and

introduces you to the programming environments of the three MattOpen tools, *Dynamic Interface*, *Requester Replacer* and *Matterhorn Macro Language*.

NOTE Some of the statements of the Matterhorn Macro Language are part of Screen Designer and require no license for MattOpen. These statements are described in Chapter 7, The Matterhorn Macro Language in the *Screen Designer Setup and Reference Guide*.

MattOpen 2.0 is part of the Matterhorn Suite, and each MattOpen component has its separate place in the Matterhorn environment. Thus, the MattOpen tools are not standalone programs, but must be used together with the Matterhorn Suite.

The manual does not describe more general issues related to the configuration and use of the other Matterhorn Suite components, since they have been described in other user guides. For a full discussion of the installation, configuration, and application of Matterhorn for Windows, turn to your Matterhorn Setup and Reference Guide**.** For information on installing and using Screen Designer, turn to your Screen Designer Setup and Reference Guide. For more information on MattWeb, turn to the MattWeb Setup and Reference Guide.

How This Book Is Organized

As mentioned, MattOpen is composed of three more or less independent components. This is reflected in the structure of this book, in which the separate parts discuss each individual component. Part 1 discusses the MattOpen Dynamic Interface, Part 2 discusses Requester Replacer, and Part 3 describes the Matterhorn macro language. The *MattOpen Developer's Guide* is organized as follows:

Introduction

Introduction to MattOpen 2.0 presents the three development tools of MattOpen; the *MattOpen Dynamic Interface*, the *MattOpen Requester Replacer*, and the *Matterhorn Macro Language*. You will learn to install and enable the tools of MattOpen.

Part 1: MattOpen Dynamic Interface

Part 1 discusses the application and operation of the MattOpen Dynamic Interface and introduces you to the programming environment of the Dynamic Interface. It presents the functions that form the interface and describes how to use them in your application development. Also, the manual discusses some of the considerations which have to be observed when using and developing applications for MattOpen Dynamic Interface. Finally, a series of examples may serve as inspiration when using MattOpen Dynamic Interface.

Chapter 1, MattOpen Dynamic Interface introduces the MattOpen Dynamic Interface and explains its overall purpose and application.

Chapter 2, Programming Environment presents the programming environment of the MattOpen Dynamic Interface.

Chapter 3, Functions and Options lists and describes all functions that make up the MattOpen dynamic link library, MattOpen DLL.

Chapter 4, Using Dynamic Interface discusses some of the considerations which should be observed when developing client applications for MattOpen Dynamic

Interface and when setting up the interface on your system. The chapter also presents the MattOpen profile.

Chapter 5, Dynamic Interface Examples is a series of relevant examples which may serve as a starting point and inspiration to your work with Dynamic Interface. Sample source code is presented and commented.

Part 2: The Requester Replacer

Part 2 discusses the MattOpen Requester Replacer which allows you to redirect entire requester calls from a legacy requester to procedures in a Windows DLL. The feature must be activated from Screen Designer. You will learn to enable and use the Requester Replacer and a series of examples may serve as inspiration when using the Requester Replacer.

Chapter 6, The Requester Replacer introduces the Requester Replacer feature and explains its overall purpose and application.

Chapter 7, Using Requester Replacer explains how to enable, use and implement Requester Replacer. The feature is activated from Screen Designer.

Chapter 8, Requester Replacer Examples is a couple of relevant examples and exercises which may serve as a starting point and inspiration to your work with the Requester Replacer. Sample source code is presented and commented.

Part 3: Matterhorn Macro Language

Part 3 discusses the Matterhorn macro language which you may use to create and run macros from your legacy

applications and/or client applications. These macros may be used to set up DDE-conversations between your legacy requesters and any DDE-compatible application, call procedures in a Windows DLL, or a combination of the two. Note that we will only present those statements of the macro language that require a MattOpen license.

Chapter 9, Matterhorn Macro Language introduces the Matterhorn macro language which enables you to create and insert macros in your legacy applications.

Chapter 10, DDE Statements presents the syntax and application of the DDE statements in the Matterhorn macro language.

Chapter 11, The CallDLL Statement presents the syntax and application of the CallDLL statement of the Matterhorn macro language.

Chapter 12, Using the Matterhorn Macro Language describes the process of creating and associating macros in Screen Designer.

Chapter 13, Macro Examples is a series of relevant examples which may serve as a starting point and inspiration when creating macros with the Matterhorn macro language.

Introduction to MattOpen 2.0

MattOpen is a group of powerful application development tools that is designed to complement Matterhorn for Windows, Screen Designer and MattWeb. The tools, *MattOpen Dynamic Interface*, *Requester Replacer* and *Matterhorn Macro Language* will make your Tandem system as open as it will ever get.

The introduction is organized as follows:

- Introducing MattOpen 2.0
- The Components of MattOpen 2.0
- Requirements for MattOpen 2.0
- Installing MattOpen 2.0

Introducing MattOpen 2.0

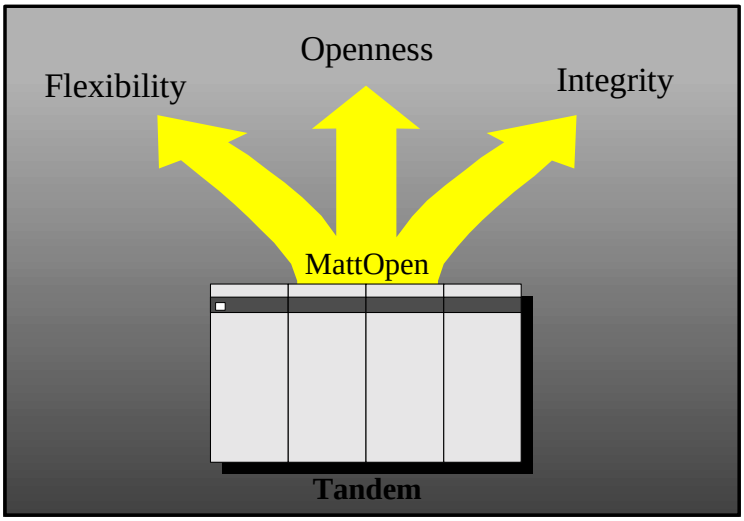
MattOpen 2.0 is composed of three independent development tools; *MattOpen Dynamic Interface*, *Requester Replacer* and *Matterhorn Macro Language*. Using the tools of MattOpen 2.0 you may:

- Access and control legacy Pathway applications on the Tandem from newer client applications on the PC.
- Redirect requester calls from legacy Pathway applications to procedures in a Windows DLL. In effect, you *replace* an entire requester with a DLL. The feature enables you to create and tailor new powerful applications without risking past investments.
- Create macros in order to call procedures in a Windows DLL from legacy Pathway applications. The DLL is called during the execution of a macro and the feature extends the functionality of the current requester, enabling you for instance to retrieve values from a remote database or validate contents of fields.
- Create and run macros in order to set up DDE conversations between your legacy Pathway applications and any DDE-compatible application.

In short, the tools of MattOpen 2.0 provide gateways to and from legacy Pathway applications. Using MattOpen, Tandem customers may continue to create and add faster and more powerful client applications in programming languages like Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or Borland Pascal and integrate them with their existing Pathway applications. Your legacy Pathway system

may be integrated with new PC-based applications with a minimum of investment and risk.

Note that MattOpen 2.0 has been developed for Matterhorn 4.0.



Openness Means Flexibility

MattOpen means openness, and openness provides flexibility. Using the tools of MattOpen you may combine your legacy Pathway applications with new PC-based applications to form new and powerful systems with a minimum of investments and absolutely no conversion on the Tandem side. Your current software investments are fully protected. Who wants to throw 50 man years down the drain?

MattOpen is particularly desirable if your company develops its subsystems on an ongoing basis. With MattOpen, the consistency and performance level of the entire system is secured. Your Tandem developers do not have to rewrite or

rethink entire Pathway application systems in new programming languages to secure optimum performance or access to the most recent technologies in the client/server area. Using MattOpen and the Matterhorn Suite they can use the best from either side and combine it into new powerful solutions.

An Example

Take the following example: A company's entire database system was originally written as a series of subsystems in Screen Cobol. In order to run their existing Pathway applications in a client/server environment the company has purchased Matterhorn for Windows. In addition, the applications has been enhanced with a new updated GUI as well as other advanced desktop features by means of Screen Designer.

But the company wanted more. The system architects decided to add new subsystems in order to extend the company database, only these would be written in a faster and more powerful Windows programming environment. At this point, the company was left with two options; either the programmers of the company had to rewrite the *entire* legacy system in the new language - (a tiresome job) - or the company would consider MattOpen, which would secure the integration of the new system with the old.

The company chose the latter option and thus gained the possibility of developing new powerful PC-based applications and at the same time fully utilize the current Pathway applications. In this book we will provide many more examples like this one to show you the prospects of MattOpen.

Note that the chapters 5, 8, and 13 provide more MattOpen examples.

16- and 32-bit MattOpen

The components of MattOpen support both 16- and 32-bit operating systems. Consequently, MattOpen supports both the 16- and 32-bit versions of Matterhorn.

Note that MattOpen 2.0 has been developed for Matterhorn 4.0.

The Matterhorn Suite

Before we go on to explore MattOpen in detail, here is a brief introduction to the rest of the Matterhorn Suite. The suite is comprised of the following four software products: *Matterhorn for Windows*, *Screen Designer*, *MattWeb*, and *MattOpen*.

Matterhorn for Windows

Matterhorn for Windows forms the core of the Matterhorn Suite. Matterhorn enables you to access your legacy Pathway applications from the PC and run them as client/server-based applications.

Matterhorn is easy to install and configure; both on Tandem and on PC and it operates equally well on both Windows 3.*xx*, Windows 95, and Windows NT platforms. Your Pathway applications will be running graphic interfaces in no time.

With Matterhorn for Windows, the maintainability of your current Tandem system is intact. No cost and time-consuming application conversions are required. Programmers do not have to rewrite one single line of code when deploying Matterhorn for Windows.

MattWeb

These days many Tandem customers are considering going on the Internet. The Internet is hot. And *intranets* is looking to become even hotter. At Skybeam we have committed ourselves to provide the Tandem world with the best Internet/intranet solution.

MattWeb is the only solution that is designed to present Pathway applications graphically on the World Wide Web. At the same time you may use MattWeb to establish your own intranet.

MattWeb fully supports requesters which has been designed with Screen Designer. At runtime, each screen element will be presented on the remote user's monitor by Java applets.

With MattWeb and Screen Designer in unison you may create *multiple mode applications*. Different versions of the same application can be run simultaneously on the Internet and your own network.

Screen Designer and MattWeb offer the same kind of flexibility to your company internally. You are not restricted to settle for a permanent client interface. For instance, one third of the users in your company may run requesters as Matterhorn sessions; another third may run the same requesters on the intranet via MattWeb and a browser; and the last third may run the original 6530/3270-sessions. With Screen Designer and MattWeb you may easily switch from one client interface to another.

For further information on MattWeb, turn to the MattWeb Setup and Reference Guide.

Figure 1.2: *A Tandem application running on the Word Wide Web.*

The browser is Netscape Navigator.

Screen Designer

Using Screen Designer, you may customize requesters on many levels. With Screen Designer you may make cosmetic changes to the interface such as removing superfluous fields, grouping selected fields, and adding new text to make the screen more intuitive to the user. You may insert backgrounds such as wallpapers and gradients and set up multimedia objects to be played when the user runs the requesters. You may create and associate hints with each object. A hint is a guiding text that explains the purpose of that particular screen object.

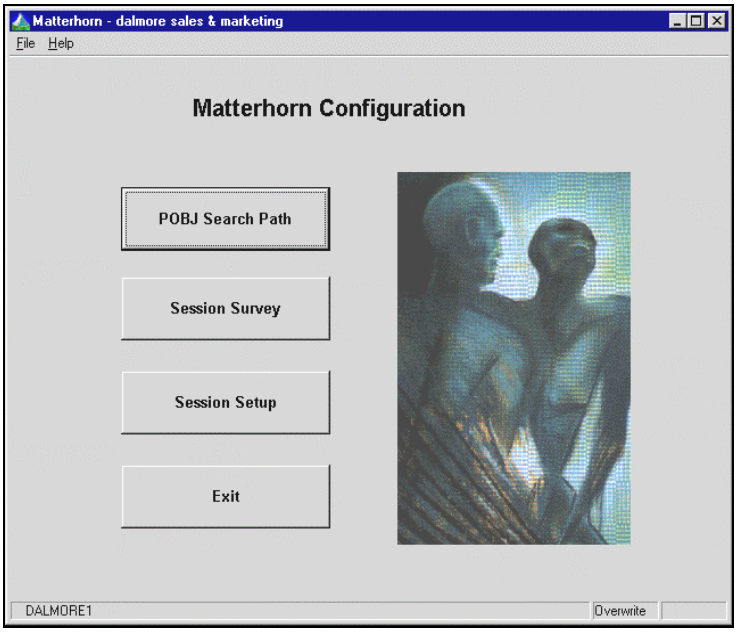


Figure I.1: *Matterhorn Screen Designer allows you to customize and optimize your current Tandem system.*

On a more functional level you may create and associate push buttons with function keys, macros, pictures, or multimedia clips. You may create macros and associate them with buttons, edit fields, pictures, and multimedia. You may create and associate selection lists and entry histories with edit fields. In addition, you may map Tandem keys to PC keys.

A new powerful feature in Screen Designer 4.0 is the ability to create *tabbed notebooks*. Tabbed notebooks enable you to organize the objects of a requester on separate tabs, just like most users know them from their Windows applications.

Screen Designer allows you to customize legacy application screens for different groups of users or individual users. Using the same screen as a basis, you may create one screen for your

bookkeeping department, another screen for the finance department and so on. Each requester screen will reflect how the current department operates.

Using Screen Designer, all the changes you may make to your legacy Pathway applications will make functionality more discoverable and common tasks simpler and more efficient. Thus, Screen Designer increases end-user productivity. Screen Designer is detailed in Screen Designer Setup and Reference Guide.

Part of Screen Designer is the Data Explorer productivity tool which provides a fast and easy way of monitoring and maintaining your Screen Designer databases on a large scale. Using Data Explorer you may create multiple requester layouts on the fly, preview your screens, and drag and drop selected objects from one requester to another and from one database to another. In addition you may drag and drop entire requesters from one database to another.

The Components of MattOpen 2.0

MattOpen consists of the following three components:

- MattOpen Dynamic Interface
- MattOpen Requester Replacer
- Matterhorn Macro Language

Each of these components are presented below and thoroughly described in Part 1, 2 and 3, respectively.

MattOpen Dynamic Interface

Use the MattOpen Dynamic Interface to access and control legacy Pathway applications from newer client applications on the PC. Note that the Dynamic Interface has been developed for Matterhorn 4.0. At the core of the MattOpen Dynamic Interface you find the MattOpen DLL, a dynamic link library which establishes and maintains the connection between the PC-based applications and your legacy Pathway applications.

MattOpen Dynamic Interface is detailed in Part 1 of this book.

MattOpen Requester Replacer

Using the MattOpen Requester Replacer, you may redirect entire requester calls from the current Pathway requester to procedures in a Windows DLL. In effect, the DLL replaces the requester, hence the name. The feature opens up the Tandem system to the powerful and flexible DLL-technology. Requester Replacer is a MattOpen feature, but you use Screen Designer to define your Requester Replaces.

MattOpen Requester Replacer is detailed in Part 2 of this book.

Matterhorn Macro Language

Use the Matterhorn macro language to create macros and integrate them with your Matterhorn sessions. You may use macros for instance to speed up and simplify administrative tasks or exchange data with other applications that support DDE or DLL. The Matterhorn macro language provides you with several related features. You may:

- create macros and run them from your legacy Pathway applications in order to set up dynamic-data

exchange (DDE) conversations with other applications that support DDE.

- create macros in the macro language of the client application to call your legacy Pathway applications.
- create and run macros that calls a Windows DLL.
- create macros that combine DLL and DDE technology.

Matterhorn may operate both as DDE-client and DDE-server. The Matterhorn macro language is detailed in Part 3 of this book.

Combining the Components of MattOpen

Each component of MattOpen can be used alone for its specific purpose or combined with the other tools as you see fit. Chapters 1, 6, and 9 provide examples to illustrate how the tools may be combined.

Requirements for MattOpen

The tools of MattOpen 2.0 require that you deploy Matterhorn 4.0 for Windows and Screen Designer. In addition you must use development environments that support DLL.

Installing MattOpen 2.0

To Install MattOpen:

1. Insert the Matterhorn Suite CD-ROM in the CD-ROM drive. Select **Run** from the **Start** menu, locate the CD-ROM and launch the Setup.exe file.

2. In the opening screen, select MattOpen. MattOpen Installation will prepare the InstallShield Wizard, which will guide you through the process.

3. Make sure to install MattOpen in your current Matterhorn folder (Matthorn). The wizard installs the 16- and 32-bit versions of the Dynamic Interface (Mattop16.dll and Mattop32.dll) sample include and header files, and various demos that illustrate the functionality of each component of MattOpen 2.0. The demos are described in detail in Chapters 5, 8 and 12.

   In your Matterhorn group folder, the dynamic link libraries are installed in the Matt16 and Matt32 subfolders. Header and include files are installed in the Include subfolder, and the demos are installed in the Demos subfolder. The demos require that you do not change the default names of the folders in the folder structure.

All the files of MattOpen 2.0 are detailed in Chapter 1 *MattOpen Dynamic Interface*, Chapter 6, *The Requester Replacer*, Chapter 10, *DDE Statements*, and Chapter 11, *The CallDLL Statement*.

Enabling Requester Replacer

The Requester Replacer option is an integral part of Screen Designer. If you have already installed Screen Designer you merely have to transfer the new license file, LICENSE, from your MattOpen diskette to your Tandem.

The next time you launch Screen Designer, the Requester Replacer option will be available from the **Define** menu.

Enabling the Matterhorn Macro Language

The Matterhorn macro language is an integral part of Screen Designer, but you cannot use the statements which are used DDE and DLL without a valid MattOpen license. The remaining statements are described in Chapter 7 of the *Screen Designer Setup and Reference Guide*.

If you have already installed Screen Designer you merely have to transfer the new license file, LICENSE, from your MattOpen diskette to your Tandem. The next time you launch Screen Designer, you will be able to create macros containing the various DDE- and DLL statements.

Part 1

The MattOpen Dynamic Interface

Part 1 introduces you to the *MattOpen Dynamic Interface*. The manual provides an overview of Dynamic Interface features and operation, and introduces you to its programming environment. It also presents the functions that form the interface and describes how to use them in your application development. We also discuss some of the considerations which have to be observed when using and developing applications for MattOpen Dynamic Interface. Finally, we present a series of demos and examples to show you how the Dynamic Interface may be used for specific purposes.

Chapter 1:
MattOpen Dynamic Interface

Chapter 1 introduces the MattOpen Dynamic Interface which allows you to access and control legacy Pathway applications from newer client applications on the PC.

The chapter is organized as follows:

- MattOpen Dynamic Interface
- Files of the MattOpen Dynamic Interface
- How Does MattOpen Dynamic Interface Work

MattOpen Dynamic Interface

The MattOpen Dynamic Interface allows PC-based client applications to access Pathway requesters on the Tandem. The Dynamic Interface is the mediator, which enables the client application to control a legacy Pathway application running under Matterhorn. In this way, requesters that have been configured to be executed by Matterhorn (so-called Matterhorn sessions) may be called from the client.

At the core of the Dynamic Interface you find the MattOpen DLL, a dynamic link library containing a series of functions which establish and maintain the connection between the PC-based client application and the requesters on the Tandem.

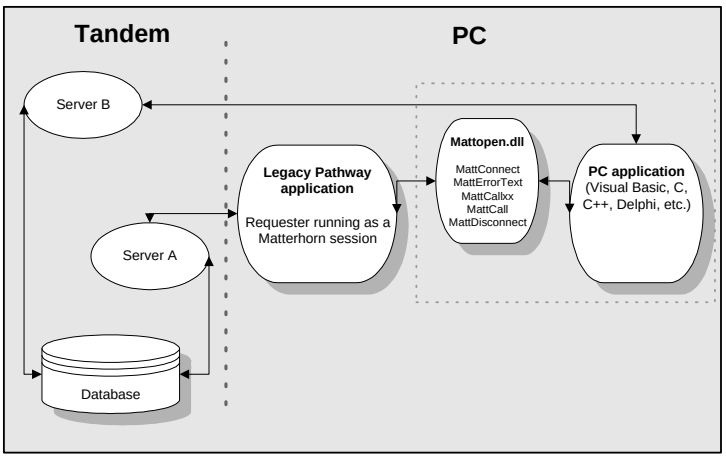


Figure 1.1: *The MattOpen Dynamic Interface establishes and maintains the connection between a client application and a legacy Pathway application running under Matterhorn.*

More figuratively, the Dynamic Interface forms a gateway through which you may establish and maintain a connection

between your new PC-based client applications and your legacy Pathway applications on the Tandem.

Securing Past and Future Software Investments

With the MattOpen Dynamic Interface, Skybeam Management Ltd. has responded to an increasing demand of its Matterhorn customers - namely the wish to develop new PC-based applications and at the same time be able to execute and exchange data with their original Pathway applications.

With MattOpen Dynamic Interface you may continue to create and add faster and more powerful client applications in programming languages like Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or Borland Pascal.

MattOpen Dynamic Interface is particularly desirable if your company develops its subsystems on an ongoing basis. With MattOpen Dynamic Interface, the consistency and performance level of your entire system is secured.

Mattop16.dll & Mattop32.dll - Dynamic Link Libraries

As mentioned, the MattOpen DLL establishes and maintains the connection between the PC-based client application and the requesters on the Tandem.

The functions exist in versions for both 16- and 32-bit Matterhorn, Mattop16.dll and Mattop32.dll, respectively. All functions are thoroughly described in Chapter 3, Functions and Options.

You may develop your applications in any programming language that supports DLL, such as Microsoft Visual C++, Microsoft Visual Basic, Borland C++ or Borland Pascal. Remember to conform the header file to the chosen

programming language. Read more about using MattOpen Dynamic Interface in Chapter 4, Using Dynamic Interface.

Files of the Dynamic Interface

The following sections present the files which are part of your MattOpen Dynamic Interface package; the DLLs, the demos and their associated support files. The files are presented under headings which reflect the subfolder in which they are stored.

The Matt16 Folder

The Matt16 folder is part of your current Matterhorn folder structure. It contains all the files for the 16-bit version of the Matterhorn Client. When you install MattOpen Dynamic Interface, the following file is copied to the folder:

Mattop16.dll	contains all functions developed for the 16-bit version of Dynamic Interface.

There are no formal demands as to the location of the DLL, but take note of the standard top-down search order when a DLL is called (see Figure 1.2).

The Matt32 Folder

The Matt32 folder is part of your current Matterhorn folder structure. It contains all the files for the 32-bit version of the Matterhorn Client. When you install MattOpen Dynamic Interface, the following file is copied to the folder:

Mattop32.dll contains all functions developed for the 32-bit version of the Dynamic Interface (see Figure 1.2).

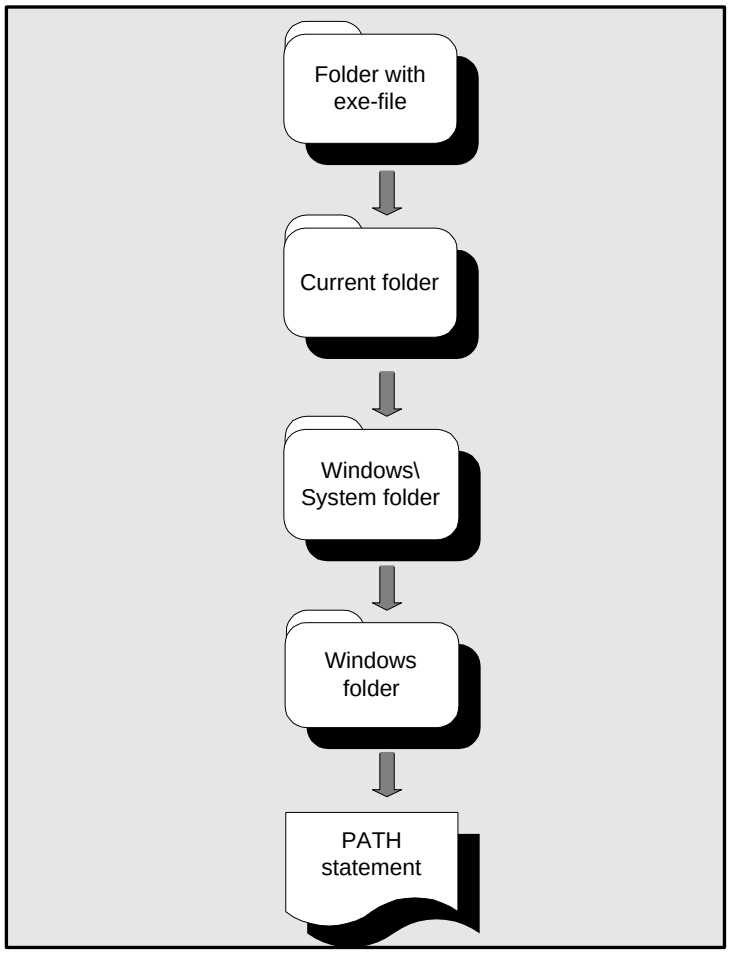


Figure 1.2: *When installing the MattOpen DLL, take note of the dynamic link library search order depicted above.*

The Include Folder

The Include folder contains three subfolders: Delphi, Msvc and Vb.

The Delphi Subfolder

This folder stores an include file for Borland Delphi.

Mattopen.pas Include file. The file lists all functions developed for MattOpen Dynamic Interface. Use this file when developing new Borland Delphi applications for the Dynamic Interface.

The Msvc Subfolder

This folder stores a header file for Microsoft Visual C++.

Mattopen.h Header file. The file lists all functions developed for MattOpen Dynamic Interface. Use this file when developing new Microsoft Visual C++ applications for the Dynamic Interface.

The Vb Subfolder

This folder stores include files for 16 and 32-bit Visual Basic.

Mattop16.inc Lists all functions developed for MattOpen Dynamic Interface. Use this file when developing new 16-bit Visual Basic applications for the Dynamic Interface.

Mattop32.inc	Lists all functions developed for MattOpen Dynamic Interface. Use this file when developing new 32-bit Visual Basic applications for the Dynamic Interface.

The Demos/Dynintf Subfolder

The Demos / Dynintf subfolder contains three subfolders: Bc16, Vb16 and Vb32. These folders contain various MattOpen demos and their support files.

The Bc16 Subfolder

The Bc16 subfolder stores a demo application written in 16-bit Borland C++ 4.5. The folder contains all support files for the demo and a Borland C++ header file:

Mattdemo.exe	The executable file of the demo. The program is thoroughly discussed in Chapter 5, Dynamic Interface Examples.
Mattopen.h	is the Dynamic Interface header file. The file lists all functions developed for MattOpen Dynamic Interface. Use this file when developing new Borland C++ applications for the Dynamic Interface.
Mattdemo.cpp	The source code of the Borland C++ demo. The source code is commented in Chapter 5, Dynamic Interface Examples.
Mattop16.ini	The MattOpen profile used by the program. The profile provides the client application with information relevant to the execution of the current Matterhorn session.

Mattdemo.def Borland C++ support file.

Mattop16.lib The MattOpen C++ import library.

Mattdemo.res Borland C++ resource file.

The Vb16 Subfolder

The Vb16 subfolder stores a demo application written in 16-bit Visual Basic. The folder contains all support files for the demo and a 16-bit Visual Basic include file:

Mattvb16.exe The executable file of the demo. The program is thoroughly discussed in Chapter 5, Dynamic Interface Examples.

Mattop16.ini The MattOpen profile used by the program.

Mattsdif.frm The Visual Basic source code file.

Mattvb16.vbp The Visual Basic project file.

Mattopen.frx Visual Basic support file.

The Vb32 Subfolder

The Vb32 subfolder stores a demo application written in 32-bit Visual Basic. The folder contains the following files:

Mattvb32.exe The executable file of the demo. The program is thoroughly discussed in Chapter 5, Dynamic Interface Examples.

Mattop32.ini	The MattOpen profile used by the program.
Mattsdif.frm	The Visual Basic source code file.
Mattvb32.vbp	The Visual Basic project file.
Mattopen.frx	Visual Basic support file.

How Does Dynamic Interface Work

As mentioned in the previous section, the link between the client application and a Pathway requester executed by Matterhorn is established and maintained by a series of functions in the MattOpen DLL. These functions will establish connections, call the requesters with optional linkage parameters, exchange data, and display error messages triggered by the requesters.

The operation flow is depicted in Figure 1.3 next page. As the figure suggests, the MattOpen DLL is to be incorporated into your PC-based client application by you placing calls to the various functions in the application source code. In Chapter 4, *Using Dynamic Interface* we line out a course of action which you may follow when you develop applications for Dynamic Interface and explains how to customize the connection between a PC-based application and a Matterhorn session.

Functions of the DLL

The MattOpen DLL contains the following functions:

MattConnect is used to connect the client application and Mattwin.exe (Matterhorn). When it is called, the function will launch Matterhorn, which will then be prepared to

execute and control the requester with valid configuration values.

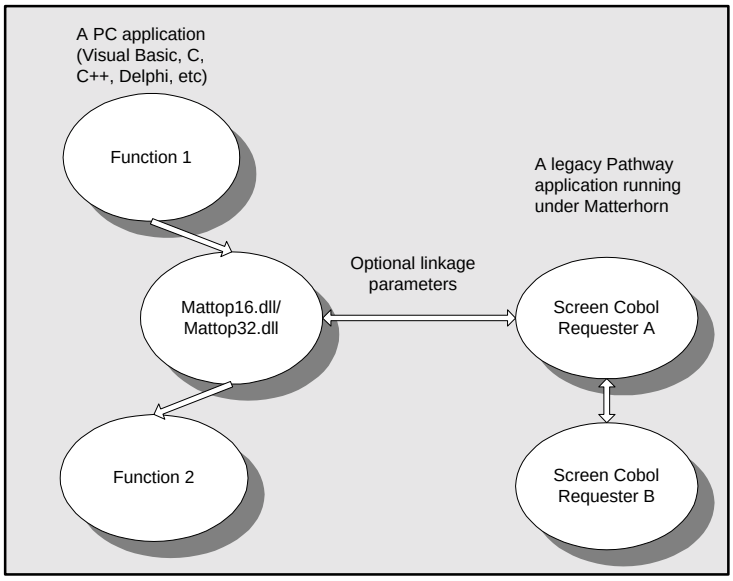


Figure 1.3: *Operation of the MattOpen Dynamic Interface.*

MattErrorText obtains the descriptive text associated with an error message returned from a requester executed by Matterhorn.

MattCall_0 makes a call to the requester without any linkage section parameters. Note that MattCall functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

MattCall_1 makes a call to the requester with one linkage section parameter.

MattCall_2 makes a call to the requester with two linkage section parameters.

MattCall_3 makes a call to the requester with three linkage section parameters.

MattCall_4 makes a call to the requester with four linkage section parameters.

MattCall is a dynamic version of the functions MattCall_0 through MattCall_4. With the function you may specify the number of parameters at runtime. Use the function when activating requesters with more than four linkage section parameters.

MattDisconnect terminates the connection between the client application and Matterhorn. To disconnect, exit your requester as you would normally exit it.

Integrating Dynamic Interface

In this section we present and comment a series of illustrations which are intended to explain how Matterhorn for Windows and the Dynamic Interface are implemented so that a legacy application - without any source conversion or prior modification - and a new PC-based application may be integrated.

At a major retailer it was decided to enhance the present booking and order system with and advanced customer and marketing module. The new module was developed in Borland Delphi. At the moment, the backbone of the system consisted of a customer file and an order file (see Figure 1.4). Now the company wanted to optimize their marketing efforts in terms of more efficient features for planning, control and follow-up on customers and prospects. For this reason they

bought a separate module designed specifically for these purposes.

The original order file were to remain a Scobol application. It provided the required functionality and no one considered altering it. Furthermore, the company still wanted to take advantage of Tandems' powerful server side.

Figure 1.4 below depicts the old system where both customers and orders were being handled on the Tandem.

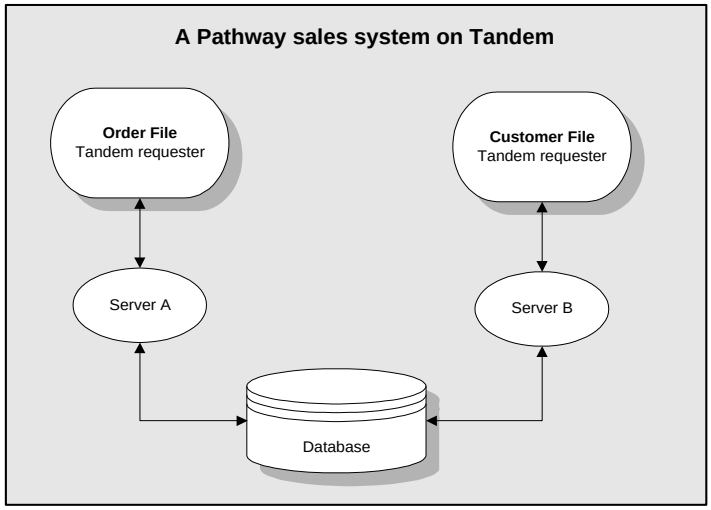


Figure 1.4: *In the old system, both orders and customers were handled on the Tandem.*

In Figure 1.5 Matterhorn for Windows has been implemented: The order file is now being handled by Matterhorn on the PC, and the new customer and marketing module has been installed, also on the PC. As seen, there is no way the customer file has access to the order file and vice versa.

The next stage is depicted in Figure 1.6: The Dynamic Interface has been introduced. Relevant calls have been placed to the

order file via the MattOpen DLL. A menu option, **Order File** has been inserted: The user simply has to click this button in order to get to the order file.

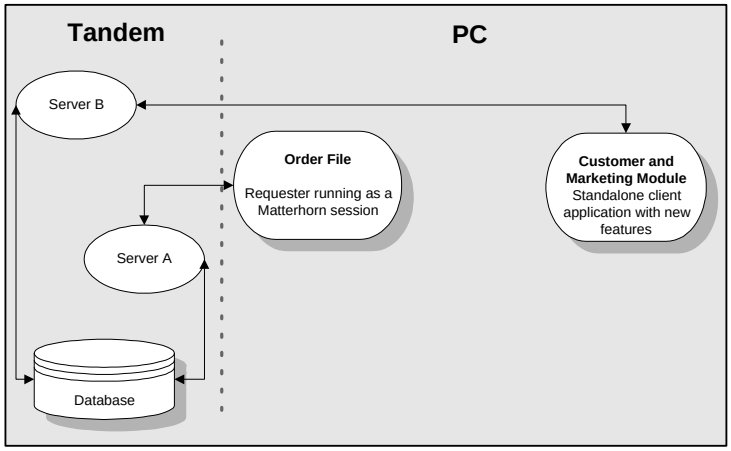


Figure 1..5: *The order file is now being handled by Matterhorn on the PC and a new customer and marketing module has been installed, also on the PC. There is no link between the customer file and the order file.*

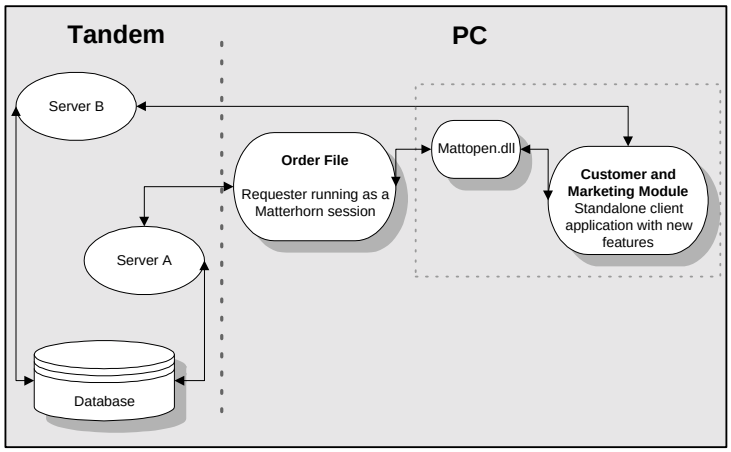


Figure 1.6: *The Dynamic Interface allows the user to access and control the order file from the customer and marketing module.*

In addition, several statistical features on the PC use data from the Tandem. These data are retrieved using MattOpen calls. In Chapter 5, Dynamic Interface Examples we will take a closer look at some of these features and describe how the MattOpen calls are implemented.

Combining Dynamic Interface with Requester Replacer

You may combine the MattOpen Dynamic Interface and the Requester Replacer feature. Let's imagine that your company is currently using the Dynamic Interface for integrating new PC-based applications with your legacy Pathway applications, and you wish to replace a certain requester with a DLL for providing better printing facilities. In this case you would define a Requester Replace in Screen Designer, making any call to this requester point to a DLL instead. At runtime, any call for the requester in question will be redirected to the DLL.

Combining Dynamic Interface with Macros

You may combine the MattOpen Dynamic Interface with macros. Imagine a situation where your system has already been set up to use the Dynamic Interface. Old and new has been successfully integrated. Now you wish to extend a legacy requester in the system, for instance with features for validating the contents of certain fields. In this case, you would write a macro in Screen Designer which makes the call to the DLL that validates the contents of the fields, and then associate the macro with a button or another screen object. At runtime you would make a call to the requester using the Dynamic Interface, and then have the contents of specific edit fields validated by activating the macro.

Chapter 2:
Programming Environment

Chapter 2 describes the programming environment of the MattOpen Dynamic Interface. The section outlines the format, notation, and naming conventions used with the Dynamic Interface's function calls and constants.

The chapter is organized as follows:

- Programming Languages
- Function Call Format
- Notation Conventions
- Constant Names

Programming Languages

You may choose to develop your applications in any programming language that supports DLL, for instance Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or Borland Pascal.

Function Call Format

All functions are in the same format. This section uses the function *MattConnect* as a basis, describing the format of all functions:

Function Name

MattConnect

The function name used when calling the function from the C language.

Name in DLL

MATTCONN

The function import name used when loading the DLL.

Description

A description of the function.

MattConnect is used to establish a connection between the client application and Mattwin.exe (Matterhorn). When called, the function will launch Matterhorn,

which will then be prepared to execute and control the requester in question. See Chapter 3, **Functions and Options** for a description of all MattOpen functions.

Syntax

The function has the following syntax:

```
extern "C" pascal short int MattConnect
       (short int *fpwMHnd, char *fpacSection,
       char *fpacProfile);
```

The syntax describes the parameters (or fields) and gives the appropriate type of each. The data type of the return value is identified by the field to the left of the function call. If a parameter is an input parameter, the function call fills in the value upon successful completion. If a parameter is an input and output parameter, the application must supply the value, but the function call updates it upon successful completion.

Parameters

A full description of each parameter (or field). See the section *Notation Conventions* for more information about the notation of parameters.

MattConnect takes three parameters, an options handle (for controlling the Matterhorn session), a section name (for identifying the Matterhorn session), and a filename (for identifying the MattOpen profile):

fpwMHnd
A handle used for the communication with Mattwin.exe.

fpacSection
A section in the MattOpen profile pointing to the Matterhorn session. Each section must contain information about where to find the Matterhorn executable file, Mattwin.exe, and the Matterhorn profile. Each section name must be unique. Only one section, i.e. service can be active at a time.

fpacProfile
The name of the MattOpen profile. This profile contains the section(s) indicated by *fpacSection*.

Return Values

The return value is zero if MattConnect fails to establish connection. If connection is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

Hungarian Notation

Hungarian notations are rules that define how to create names that indicate both the purpose and data type of an item. All parameters and variables in this manual conform to the notation convention shown in this section. These naming conventions are used in this manual to help you identify the purpose and type the function parameter and fields.

All parameter and field names consist of up to three elements; a *prefix*, a *base type* and a *qualifier*. For example the parameter, *fpacProfile* has the following elements:

- fp is the prefix
- c, character, is the base type
- *Profile*, is the qualifier

The prefix, written in lowercase letters, specifies additional information about the item, for instance whether it is a pointer, an array, or so on. The base type, also written in lowercase letters, identifies the data type of the item. The qualifier, a short word or phrase written with the first letter of each word in upper-case letters, specifies the purpose of the item. Standard prefixes are:

Prefix:	Description:
p	pointer
fp	far pointer
a	array
fpac	far pointer array

Standard base types are:

Base type:	Type/Descrip ion:
c	char
l	long
n	int
s	structure
u	unsigned
v	void

Constant Names

A constant name is a descriptive name for a numeric value used with the MattOpen Dynamic Interface functions. If you use constant names in your code and the value of the constant changes in the Dynamic Interface header file, you will not need to make the change throughout the program.

All constant names are written in upper-case letters and are prefixed with the string “MO_”. The rest of the constant name identifies the meaning of the constant.

Chapter 3:
Functions and Options

This chapter lists and describes all functions that make up the dynamic link libraries, Mattop16.dll and Mattop32.dll. The DLLs are stored in the Matt16 and Matt32 subfolder.

The chapter is organized as follows:

- Functions of the Dynamic Interface
- MattOpen Error Codes

Functions of the Dynamic Interface

This section lists all the functions that make up the MattOpen DLL, and gives a short description of each program. All functions are included in both Mattop16.dll and Mattop32.dll.

You may develop your applications in any programming language that supports DLL, like Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or Borland Pascal. Remember to conform the Dynamic Interface header file to the chosen programming language.

To sum up, the following functions reside in MattOpen DLL.

- MattConnect
- MattErrorText
- MattCall_0
- MattCall_1
- MattCall_2
- MattCall_3
- MattCall_4
- MattCall
- MattDisconnect

MattConnect

Function Name

MattConnect

Name in DLL

MATTCONN

Description

MattConnect is used to connect the client application and Mattwin.exe (Matterhorn). When called, the function will launch Matterhorn, which will then be prepared to execute and control the requester with valid configuration values.

Syntax

The function has the following syntax:

```
extern "C" pascal short int MattConnect
       (short int *fpwMHnd, char *fpacSection,
       char  fpacProfile);
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

fpacSection
A section in the MattOpen profile pointing to the Matterhorn session. Each section must contain information about where to find the Matterhorn

executable file, Mattwin.exe, and the Matterhorn profile. Each section name must be unique.

fpacProfile
The name of the MattOpen profile. This profile contains the section(s) indicated by *fpacSection*.

Return Value

The return value is zero if MattConnect fails to establish connection. If connection is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattErrorText

Function Name

MattErrorText

Name in DLL

MATTETXT

Description

This function obtains the descriptive text associated with an error message returned from a requester executed by Matterhorn.

Syntax

The function has the following syntax:

```
extern "C" pascal MattErrorText
       (short int fpwMHnd, char *fpacMsg, short
       int wLen);
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacMsg*
The pointer to the location to receive a textual description of the error. This parameter should point to a character buffer of the size stored in the integer pointed to by wLen, which should be at least 32 bytes long. The error text will be truncated if the buffer is too short. The maximum length of any message will not be longer than 256 bytes.

wLen
On input the pointer to the maximum number of characters the function should copy to the location specified by MattErrorText.

Return Value

The return value is zero if MattErrorText fails. If connection is successful then the return value is non-zero.

MattCall_0

Function Name

MattCall_0

Name in DLL

MATTCAL0

Description

This function will make a call to the requester without any linkage section parameters. Note that MattCall functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall_0
       (short int fpwMHnd, char *fpacUnit );
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

Return Value

The return value is zero if MattCall_0 fails. If the call is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattCall_1

Function Name

MattCall_1

Name in DLL

MATTCAL1

Description

This function will make a call to the requester with one linkage section parameter. Note that MattCall functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall_1
       (short int fpwMHnd, char *fpacUnit, void
       *fpacL1, short int wSize1 );
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

**fpacL1*
A pointer to the linkage section data. Remember to conform data delivered through this pointer to the format required by the requester. For instance, a binary field must be swapped.

wSize1
The size of the linkage section data parameter in bytes.

Return Value

The return value is zero if MattCall_1 fails. If the call is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattCall_2

Function Name

MattCall_2

Name in DLL

MATTCAL2

Description

This function will make a call to the requester with two linkage section parameters. Note that MattCall functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall_2
       (short int fpwMHnd, char *fpacUnit, void
       *fpacL1, short int wSize1, void *fpacL2,
       short int wSize2);
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

**fpacL1*
First pointer to the linkage section data. Remember to conform data delivered through this pointer to the format required by the requester. For instance, a binary field must be swapped.

wSize1
The size of the first linkage section data parameter in bytes.

**fpacL2*
Second pointer to the linkage section data.

wSize2
The size of the second linkage section data parameter in bytes.

Return Value

The return value is zero if MattCall_2 fails. If the call is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattCall_3

Function Name

MattCall_3

Name in DLL

MATTCAL3

Description

This function will make a call to the requester with three linkage section parameters. Note that MattCall functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall_3
       (short int fpwMHnd, char *fpacUnit, void
       *fpacL1, short int wSize1, void *fpacL2,
       short int wSize2, void *fpacL3, short int
       wSize3);
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

**fpacL1*
First pointer to the linkage section data. Remember to conform data delivered through this pointer to the format required by the requester. For instance, a binary field must be swapped.

wSize1
The size of the first linkage section data parameter in bytes.

**fpacL2*
Second pointer to the linkage section data.

wSize2
The size of the second linkage section data parameter in bytes.

**fpacL3*
Third pointer to the linkage section data.

wSize3
The size of the third linkage section data parameter in bytes.

Return Value

The return value is zero if MattCall_3 fails. If the call is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattCall_4

Function Name

MattCall_4

Name in DLL

MATTCAL4

Description

This function will make a call to the requester with four linkage section parameters.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall_4
       (short int fpwMHnd, char *fpacUnit, void
       *fpacL1, short int wSize1, void *fpacL2,
       short int wSize2, void *fpacL3, short int
       wSize3, void *fpacL4, short int wSize4 );
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

**fpacL1*
First pointer to the linkage section data. Remember to conform data delivered through this pointer to the format required by the requester. For instance, a binary field must be swapped.

wSize1
The size of the first linkage section data parameter in bytes.

**fpacL2*
Second pointer to the linkage section data.

wSize2
The size of the second linkage section data parameter in bytes.

**fpacL3*
Third pointer to the linkage section data.

wSize3
The size of the third linkage section data parameter in bytes.

**fpacL4*
Fourth pointer to the linkage section data.

wSize4
The size of the fourth linkage section data parameter in bytes.

Return Value

The return value is zero if MattCall_4 fails. If the call is successful then the return value is non-zero. To retrieve the error code associated with a failure, use MattErrorText.

MattCall

Function Name

MattCall

Name in DLL

MATTCALL

Description

MattCall is a dynamic version of functions MattCall_0 through MattCall_4. With this function you may specify the number of parameters at run-time. Use the function when activating requesters with more than four linkage section parameters.

Syntax

The function has the following syntax:

```
extern "C" pascal short int  MattCall
       (short int fpwMHnd, char *fpacUnit, short
       int siPCnt, void* fpapLink[], short int
       fpaiSize[] );
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

**fpacUnit*
The requester to be activated.

siPCnt
The number of linkage section parameters required by the requester.

fpapLink[]
An array of pointers to the linkage section parameters. The number of pointers must equal the number of linkage section parameters specified for siPCnt.

fpaiSize[]
The size of each linkage section parameter. The number of sizes must equal the number of linkage section parameters specified for siPCnt.

Return Value

The return value is zero if MattCall fails. If connection is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

MattDisconnect

Function Name

MattDisconnect

Name in DLL

MATTDISC

Description

This function terminates the connection between the client application and Matterhorn. To disconnect, exit the requester as you would normally.

Syntax

The function has the following syntax:

```
extern "C" pascal short int MattDisconnect (short
       int *fpwMHnd );
```

Parameters

fpwMHnd
A handle used for the communication with Mattwin.exe.

Return Value

The return value is zero if MattDisconnect fails. If disconnection is successful then the return value is non-zero. To retrieve the error code associated with a failure, use the function MattErrorText.

Dynamic Interface Error Codes

This section lists the Dynamic Interface error codes and their textual descriptions:

```
Const
#define MO_ERR_DDE  1
#define MO_ERR_APPL 2
#define MO_ERR_MATT 3
#define MO_ERR_SERV 4
#define MO_ERR_EXEC 5
#define MO_ERR_RUN  6

 #define MO_ERR_APPL_HND_INV 1
/*Handle out of bounds or free */

#define MO_ERR_APPL_ALL_USED  2
/* No more handles available */

#define MO_ERR_APPL_INTERNAL  3
/* Internal error in MattOpen*/

#define MO_ERR_APPL_SERV_USED 4
/* This server name is already used */

#define MO_ERR_APPL_SERV_BUSY 5
/* This server is already executing */

#define MO_MATT_ALLOC_WS_OVERFLOW  10
/*Trying to set too much data on call params */

#define MO_MATT_RUN_ERROR 11
/* MattWin detected a run time error */

#define MO_MATT_UNKNOWN_FUNCTION 12
/*Invalid function send to MATTWIN */

#define MO_MATT_INVALID_PARM  13 /* Invalid
       parameter */
#define MO_MATT_STOPPED  14
/* MATTWIN stopped unexpected as a task */

#define MO_MATT_POSTAPP  15
```

```
/* PostQuitMessage to MATTWIN failed */

#define MO_ERR_SERV_VERSION  20
/* Server responded with invalid version*/

#define MO_ERR_SERV_ERROR  21
/*Server responded with an error code */

 #define MO_ERR_SERV_TRUNC_REPL  22
/* Reply was truncated */
```

Chapter 4:
Using Dynamic Interface

This chapter discusses how to use the Dynamic Interface and describes some of the considerations which should be observed when developing client applications for MattOpen Dynamic Interface and when setting up the interface on your system. The chapter also presents the MattOpen profile, which must be prepared correctly when calling Matterhorn from a client application.

The chapter is organized as follows:

- Using Dynamic Interface
- Development Considerations

Using Dynamic Interface

When setting up your system to use MattOpen Dynamic Interface you must perform the steps outlined below.

- Conform the header file to the programming language of the client application.
- Prepare a MattOpen profile to point to one or several Matterhorn sessions on the PC.
- Place function calls in the source code of the client application.

Note that the chapter assumes that you have installed the Dynamic Interface (see Chapter 1, The MattOpen Dynamic Interface) and also that you have created the Matterhorn sessions you wish to call (see the Matterhorn for Windows Setup and Reference Guide).

Conforming the Header File

First, you must make sure to conform the Dynamic Interface header file to the programming language of the client application.

In the MattOpen package, we have included a series of header and include files for the most common programming languages; Borland C++ (4.5), 16- and 32-bit Visual Basic, 16- and 32-bit Visual C++, and 16- and 32-bit Borland Delphi. These files are installed in the relevant subfolders of your Matterhorn folder. If you use other programming languages than these, you must conform the header file yourself.

Preparing the MattOpen Profile

The MattOpen profile provides the client application with information relevant to the execution of the current Matterhorn session. More precisely, the profile contains information about which Matterhorn 4.0 client to execute and which Matterhorn profile to initiate. Each service in the profile is represented by a *section.* A section in the MattOpen profile has the following syntax:

```
[SectionName]
MattWin = path to a valid Matterhorn 4.0 version
MattConf = path to a valid Matterhorn profile
```

A MattOpen profile may contain as many sections as you wish, but only one section can be active at a time on the same workstation. The sample profile below contains three sections:

```
[MATTWIN]
MattWin = c:\Matthorn\Mattwin.exe
MattConf = c:\Matthorn\Mattconf.ini


[DALMORE]
MattWin = c:\Matthorn\Mattwin.exe
MattConf = c:\Matthorn\Dalmore.ini


[CUSTOMER]
MattWin = c:\Matthorn\Mattwin.exe
MattConf = c:\Matthorn\customer.ini
```

Placing Function Calls

When the header file(s) and MattOpen profile(s) are in place, you are ready to place function calls in the source code of the

client application. On the following pages we suggest a logical order in which you can place the calls:

1. MattConnect - establishing the connection
2. MattCall functions - making the relevant calls
3. MattErrorText - debugging
4. MattDisconnect - closing the connection.

MattConnect - Establishing the Connection

The first function to use is *MattConnect*. The function establishes the connection between the client application and the Matterhorn session. Technically, its prime objective is to locate the Matterhorn client and the Matterhorn session and return a handle which is used for the communication. The function takes the following three parameters:

In the following function call:

```
Result = MattConnect (Hndl, “Customer”,
       “Dalmore.ini”);
```

MattConnect looks for the Customer section in the MattOpen profile, Dalmore.ini.

MattCall - Making Calls

When the MattConnect function has been integrated, you proceed to place calls to the different MattCall functions as you see fit. For this purpose MattOpen provides functions for different numbers of linkage section parameters:

MattCall_0	No linkage section parameters.
MattCall_1	One linkage section parameter.
MattCall_2	Two linkage section parameters.
MattCall_3	Three linkage section parameters.
MattCall_4	Four linkage section parameters.
MattCall	Dynamic version. The user specifies the number of parameters at runtime.

For each call, you will have to indicate the handle to use, the requester to activate, and the linkage section parameters along with their sizes. Note that the functions are waited calls. The functions will return only when the requester does an exit program, stops running, or terminates due to run-time error.

In the following function call:

```
Result = MattCall_0(Hndl, “Custom1”);
```

the MattCall_0 function calls the requester “Custom1”.

In the following function call:

```
Result = MattCall_2 (Hndl, “Custom1”, &Customer,
       sizeof(Customer), &Order, sizeof(Order));
```

the MattCall_2 calls the requester “Custom1” with two optional linkage section parameters, the structures “Customer” and “Order”.

In the following function call:

```
Result = MattCall (Hndl,"Custom1", 10, &Customers,
       &CustomerSizes);
```

the MattCall function calls the requester "Custom1" with 10 (ten) parameters contained in the array "Customers". The "CustomerSizes" array contains the size of each Customer structure.

MattDisconnect - Closing the Connection

When all functions calls have been entered correctly, the connection is terminated by the MattDisconnect.

In the following function call:

```
Result = MattDisconnect(Hndl);
```

MattDisconnect terminates the connection identified by the handle "Hndl".

MattErrorText - Debugging

If you experience problems you can include the function MattErrorText to assist you in debugging.

In the following function call:

```
Result = MattErrorText (Hndl, Msg, sizeof(Msg));
```

MattErrorText obtains the descriptive text associated with an error message returned from the Matterhorn session identified by the handle "Hndl".

Development Considerations

This section discusses the considerations that are specific to developing and compiling MattOpen Dynamic Interface applications. The section presents the features for data formatting and conversion provided with MattOpen and introduces the steps involved when compiling an application.

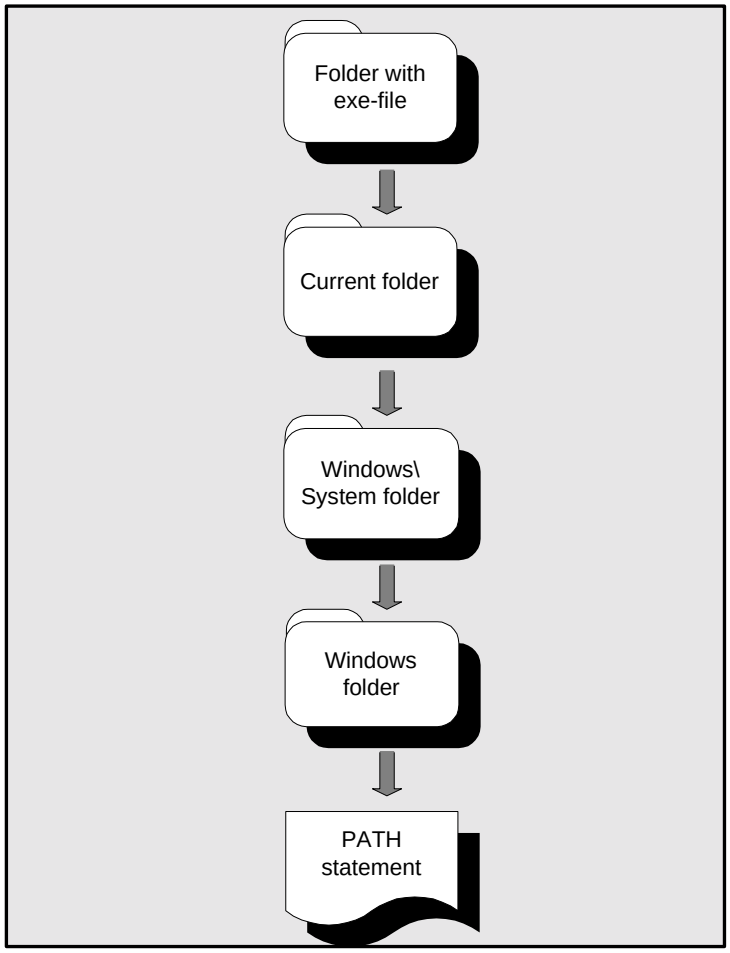


Figure 4.1: *When installing the MattOpen DLL, take note of the dynamic link library search order depicted above.*

The Location of Mattop16.dll & Mattop32.dll

The dynamic link libraries, Mattop16.dll and Mattop32.dll, are installed in the Matt16 and Matt32 subfolders, respectively. There are no formal demands as to the location of DLL, but you should take note of the standard top-down search order when a DLL is called (see Figure 4.1).

Formatting and Converting Data

It is important to realize that data on the Tandem do not match data on the PC. MattOpen Dynamic Interface provides the following API functions to swap integers back and forth between the PC and the Tandem.

- The MOSwapInt function converts 2-byte data.
- The MOSwapLong function converts 4-byte data.

The following types of data conversion are *not* supported by MattOpen; applications must convert these data types before calling MattOpen:

- COBOL string to C language string-trailing spaces versus terminating NULL
- Data alignment. IBM compatible PC needs no special alignment (word aligned for 2 byte integers).
- Ranges of values-C language signed integer up to 32.767 and COBOL “PIC S9(4) COMP” up to 9.999.
- National language support-7-bit substitution.

Compiling an Application

Compiling a MattOpen Dynamic Interface application containing the MattOpen import library generates a number of object files. Linking the program generates the executable file. When running the executable under Windows, the MattOpen DLL is dynamically loaded.

Linking the MattOpen Libraries

DLLs are similar to runtime libraries except they are linked with the application, when the application is run, not when the application is linked with the linker. This method of linking a library is called *dynamic linking.* DLLs make efficient use of memory because only one copy of a library is resident in memory at any given time. No matter how many programs are using the services available through the DLL, there will still be only one copy in memory. Linking a DLL may require more system resources than linking a static library.

To link an application to a DLL, you must:

1. Perform a static link to the import library.
2. Call the import library to provide the linker with the information necessary to set up relocation tables for the functions in the DLL. These relocation tables are used to support the dynamic linking the occurs at run time.

The MattOpen Dynamic Interface libraries involved in dynamic linking are named as follows:

- Mattop16.lib / Mattop32.lib. The MattOpen import library.
- Mattop16.dll / Mattop16.dll. The MattOpen dynamic link library.

Mattop16.lib and Mattop32.lib are import libraries for the relevant DLL and must be linked to the application. The import library is generated with the ImpLib.exe utility supplied with your compiler.

Applications are linked with the MattOpen API using any linker which supports the library file format. The library (included with your MattOpen package) contains all external code and data references necessary to resolve calls to the MattOpen API.

Chapter 5:
Dynamic Interface Examples

This chapter presents a series of relevant examples which may serve as a starting point and inspiration to your work with Dynamic Interface.

The chapter is organized as follows:

- Borland C++ Demo
- 32-bit Visual Basic Demo
- Two Samples

Borland C++ Demo

As part of your MattOpen package, you will find an application written in Borland C++ (version 4.5). It is stored in the Demos\Dynintf\Bc16 subfolder and represented by the **MattOpen Borland C++** icon in the MattOpen folder.

The application is an interface to Matterhorn's own setup tool on Tandem, Matterhorn Configuration, which is used to create and edit Matterhorn sessions (see also the Matterhorn for Windows Setup and Reference Guide).

Note that the entire source code of the application is also stored in the Demos\Dynintf\Bc16 subfolder. On these pages we will merely describe the calls to the Dynamic Interface.

To launch the MattOpen Borland C++ demo, double-click the **MattOpen Borland C++** icon. The opening screen of the program is depicted in Figure 5.1 below.

Figure 5.1: *The opening screen of the MattDemo program.*

Troubleshooting the Demo

If the application will not start, right-click the icon and select Properties from the popup menu. In the Properties dialog box check that the command line contains valid references to Mattwin.exe and the MattOpen profile, Mattconf.ini. Save your changes and close the dialog box.

If the application still wont run, open the MattOpen profile, Mattop16.ini in the BC16 folder, and verify that the [MATTWIN]-section contains the correct settings. The **MattWin** entry should point to a valid version of 16-bit Matterhorn 4.0 for Windows. The **MattConf** entry should point to a valid Matterhorn profile which contains

information that allows Matterhorn Configuration to be executed as a Matterhorn session. By default, the profile contains the following two lines:

```
MattWin = c:\Matthorn\Matt16\Mattwin.exe
MattConf = c:\Matthorn\ Matt16\Mattconf.ini
```

If these settings do not apply to the location of your Matterhorn program files and support files, please adjust the MattOpen profile to your environment.

Explaining the Demo

The demo application window contains the three buttons; **Session Setup**, **POBJ Search Paths**, and **Load Sessions**.

The Session Setup Button

Clicking the **Session Setup** button takes you to the **Session Setup** window of the Matterhorn Configuration program (the requester *Mattrq04*). This window is the starting point when creating or editing Matterhorn sessions. If you enter a name in the **Session Name** field and then click the **Session Setup** button, the application takes you to the information sheet of the session where you may edit (or create) the session.

Technically, clicking the **Session Setup** button initiates a MattConnect function with the following syntax:

```
Result = MattConnect (Hndl, Section, Profile);
```

where Section is a variable containing the section name in the MattOpen profile. Mattopen.ini contains the following lines:

```
MattWin = c:\Matthorn\Matt16\Mattwin.exe
```

```
MattConf = c:\Matthorn\Matt16\Mattconf.ini
```

where Mattwin.exe is the Matterhorn Client executable file and Mattconf.ini is a Matterhorn profile pointing to a Matterhorn session which has access to the requester, *Mattrq04*. You may now navigate the **Session Setup** window - and edit or create new Matterhorn sessions. As you exit the program, the session name will be returned to the demo.

If you enter a name in the field **Session Name**, and click the **Session Setup** button, this name will serve as parameter in the function *MattCall_1*. The function call has the following syntax:

```
Result = MattCall_1 (Hndl, “Mattrq04”,
       &SessionName, sizeof(SessionName));
```

The POBJ Search Path Button

Clicking the **POBJ Search Path** button takes you to the POBJ Search Path window in the Matterhorn Configuration program. Use this window to define new server classes and enter POBJ search paths for your Matterhorn sessions.

Technically, clicking the **POBJ Search Path** button initiates call to the MattCall_0 function.

```
Result = MattCall_0 (Hndl, “Mattrq04”));
```

The Load Sessions Button

If you click this button, all available Matterhorn sessions will be retrieved and inserted in the **Available Sessions** list box in the right-hand side of the application window.

Technically, the application performs a series of RSC sends to the MattConf server and locates all session names which have been defined.

32-bit Visual Basic Demo

As part of your MattOpen package, you find an application written in 32-bit Visual Basic. It is stored in the Demos\Dynintf\Vb32 subfolder and represented by the **MattOpen VisualBasic 32** icon in the MattOpen folder.

To launch the MattOpen Visual Basic 32 demo, double-click the **MattOpen VisualBasic 32** icon. The opening screen of the program is depicted in Figure 5.2 next page.

Explaining the Demo

The purpose of the demo is to show you how various functions of the Dynamic Interface operates. The demo includes calls to all requesters accessible from the POBJ search path, including *rqzt*, and *rqxr* which we will describe on the following pages.

The Section Name Field

This field contains the section name of the MattOpen Profile. Default is MATTWIN, but you may enter the valid section name of a valid MattOpen profile. If you have installed MattOpen in the default folders, you will not have to change these options.

Figure 5.2: *The application window of the 32-bit Visual Basic MattOpen demo.*

The Profile Field

This field contains a path to a MattOpen profile. Default is a path leading to the MattOp32.ini in the Demos\Dynintf\Vb32 subfolder, which contains the MATTWIN section indicated in the **Section Name** field. You may change the path to point to another valid MattOpen profile.

The Connect Button

Click **Connect** to establish a connection to the Matterhorn Client using the contents of the fields **Section Name** and **Profile** as parameters in the MattConnect function. In this example the MattConnect statement looks like this:

```
Result = MattConnect (&Hndl, Section, Profile);
```

where Section is the section name in the MattOpen profile. Mattop32.ini contains the following lines:

```
MattWin = c:\Matthorn\Matt32\Mattwin.exe
MattConf = c:\Matthorn\Matt32\Mattconf.ini
```

where Mattwin.exe is the Matterhorn Client executable file and Mattconf.ini is a Matterhorn profile pointing to all requesters in the Matterhorn POBJ.

Increment

Clicking this button causes the number (in the field to the right of the button) to be increased by one. Technically, a call with one parameter will be made to the requester *rqzt* using the MattCall_1 function The parameter is the number in the field to the right of the button. Rqzt will then add one to this number and return the value. The MattCall_1 function has the following contents:

```
Result = MattCall_1 (Hndl, “rqzt”, &Number,
       sizeof(Number);
```

Where number is the number in the edit field.

Calculate

When you click **Calculate** the numbers in the first and second fields to the right of the button will be added, and the result will be inserted in the third field. Technically, a call with three parameters is made to the requester *rqrx* using the MattCall_3 function.

In the demo source code, the MattCall_3 function has been modified to conform to Visual Basic standards, since Visual Basic does not support typecasting. Below you see the C syntax of the call:

```
Result = MattCall_3 (Hndl, “rqrx”, &Field1,
       sizeof(Field1), &Field2, sizeof(Field2),
       &Result, sizeof(Result));
```

The Call Button

Use this button to call any requester in the POBJ search path with no optional linkage parameters.

```
Result = MattCall_0 (Hndl, Requester);
```

Disconnect

Clicking this button will terminate the connection to the Matterhorn Client. The MattDisconnect function looks like this:

```
Result = MattDisconnect(Hndl);
```

Show MattWin Errors

When Matterhorn reports any errors they will be displayed in the Errors section. If more than one error has been reported you can click the **Show MattWin Errors** button to view the entire contents of the error log.

```
Result = MattErrorText(Hndl);
```

Two Samples

In Chapter 1 we presented a case in which a company had implemented a new PC-based customer and marketing module while maintaining the original Scobol order file on the Tandem. In this section we present two statistical features of that system which require that data be retrieved from the Tandem, and describe how the retrieval is made possible using the Dynamic Interface.

In the following, we have skipped the description of the MattConnect function assuming that you are familiar with the process of using this function.

Update Priority

The first feature assigns a new priority to a customer if the total purchases of the customer exceed specific marginal values. If a customer has purchased for less than $ 2,000 then the **Priority** field will be set to D, if the total lies between $ 2,000 and $ 20,000, the field will be set to C, etc.

From a functional viewpoint, when the user clicks the button **Update Priority**, a call will be made to the order file using the MattCall_1 function. The optional linkage parameter is the Customer number. The UpdatePriority procedure then

calculates the total, assigns a priority, and returns it to the client application.

```
/* Sets Priority according to total purchase */

void UpdatePriority(long CustomerNo)
{
  long Total; /* Holds total of customer purchase */

  /* Get total of customer purchase */
  if MattCall_1( hDLL, "RQCUSTOT",
    &CustomerNo, sizeof(CustomerNo),
    &Total, sizeof(Total) )
    return;   /* Error */

  /* Set priority according to total */
  if (Total < 2000)
    SetCustomerPriority( CustomerNo, 'D' );
  else if (Total < 20000)
    SetCustomerPriority( CustomerNo, 'C' );
  else if (Total < 200000L)
    SetCustomerPriority( CustomerNo, 'B' );
  else
    SetCustomerPriority( CustomerNo, 'A' );
}
```

Get Key Figures

The second feature retrieves various customer key figures within a specific period in the order file. The dynamic function MattCall is used to call the order file. Based on the customer number and a start date and end date, the function shows four of 30 possible key figures: Total, Tracking, Revenues, and Cost.

```
/*Retrieves key figures within in a specific period*/
```

```
void GetKeyFigures(long CustomerNo, char
       *StartDate, char *EndDate)
{
 #define FIGURES 30  /*There are 30 key figures*/
 long Args[FIGURES];  /*Arguments array */
 void *pArgs[FIGURES] /*Pointers to arguments*/
 short int ArgSizes[FIGURES]; /*Sizes of the arguments*/
  int i;                    /* Iterator*/

  /* Setup pointer array and sizes */
  for(i = 0; i < FIGURES; i++)
  {
    pArgs = &Args[i];
    ArgSizes[i] = sizeof(long);
  }

  /*Set up criteria in the first 3 arguments*/
  Args[0] = CustomerNo;
  Args[1] = (void *)StartDate;
  Args[2] = (void *)EndDate;

  /* Call requester for key figures */
  if ( MattCall ( hDLL, "RQCUSNUM", FIGURES,
       pArgs, ArgSizes) )
    return;   /* Error */

  /* Show key figures */
  ShowTotal(Args[TOTAL]);
  ShowTracking(Args[TRACKING]);
  ShowRevenue(Args[REVENUE]);
  ShowCost(Args[COST]);
}
```

Part 2

The Requester Replacer

Part 2 discusses the MattOpen Requester Replacer, which allows you to redirect entire requester calls from a legacy requester to procedures in a Windows DLL. The feature must be activated from Screen Designer. You will learn to enable and use the Requester Replacer, and a couple of examples and exercises may serve as inspiration when using the Requester Replacer.

Chapter 6:
The Requester Replacer

Using the MattOpen Requester Replacer, you may redirect entire requester calls from the current requester to procedures in a Windows DLL. The feature, which must be enabled from Screen Designer, opens up the Tandem system to the powerful and flexible DLL-technology.

The chapter is organized as follows:

- Introducing Requester Replacer
- Screen Designer and Requester Replacer
- Enabling Requester Replacer
- How Does Requester Replacer Work
- Files of Requester Replacer

Introducing Requester Replacer

This chapter introduces the MattOpen Requester Replacer which allows you to redirect entire requester calls from a legacy requester to procedures in a Windows DLL.

Use Requester Replacer when you recognize that certain elements of your legacy system could be optimized using PC-based technology. With Requester Replacer you do not have to dump the entire legacy system just because certain requesters do not provide the required functionality. Simply replace the less functional requesters with DLLs as you see fit.

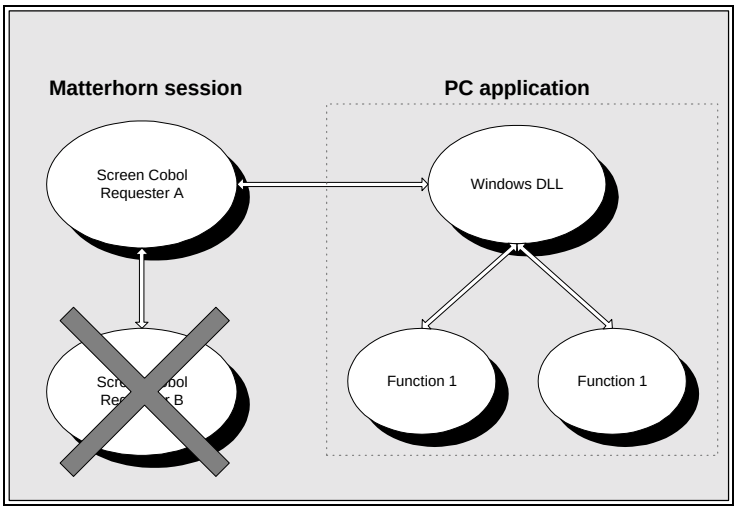


Figure 6.1: *The Requester Replacer enables you to replace an entire requester with a Windows DLL.*

Screen Designer and Requester Replacer

The Requester Replacer is an integral part of Screen Designer, which means that you must have a valid license for Screen Designer in order to use Requester Replacer. Chapter 7, Using Requester Replacer provides more information on using the feature in Screen Designer.

Enabling Requester Replacer

The Requester Replacer option is an integral part of Screen Designer. If you have already installed Screen Designer you merely have to transfer the new license file, LICENSE, from your MattOpen installation diskette to your Tandem.

The next time you launch Screen Designer, the Requester Replacer option will be available from the **Define** menu.

How Does Requester Replacer Work

You simply define in Screen Designer that a requester is to be replaced by a valid Windows DLL each time the requester is called. All variables in the working storage will be passed to the DLL and to the procedure.

In order to replace a requester you need two things:

1. A DLL with an appropriate interface.
2. A Requester Replace definition in Screen Designer.

In Chapter 7, Using Requester Replacer we will explore these issues in detail.

Combining Requester Replacer with Dynamic Interface

You may combine the MattOpen Dynamic Interface and the Requester Replacer feature. Let's imagine that your company is currently using the Dynamic Interface for integrating new PC-based applications with your legacy Pathway applications, and you wish to replace a certain requester with a DLL for providing better printing facilities. In this case you would define a Requester Replace in Screen Designer, making any call to this requester point to a DLL instead. At runtime, any call for the requester in question will be redirected to the DLL.

Files of Requester Replacer

As part of your MattOpen package you find a couple of examples and exercises which are installed in the Demos\Reqrepl subfolder. The examples have been developed in both Borland Delphi and Microsoft Visual C++ and they are detailed in Chapter 8, Requester Replacer Examples.

The Include\Delphi Subfolder

This folder stores a Borland Delphi include file.

Reqrpl.pas	Include file. The file defines the types needed for implementing Requester Replace DLLs developed in Borland Delphi.

The Include\Msvc Subfolder

This folder stores a Microsoft Visual C++ header file.

Reqrpl.h	Header file. The file defines the types needed for implementing Requester Replace DLLs developed in Microsoft Visual C++.

The Demos\Reqrepl

The Demos\Reqrepl subfolder contains the files needed for creating the sample Requester Replacer demos. The exercises are described in Chapter 8, Requester Replacer Examples.

The Reqrepl\Delphi Subfolder

This subfolder contains the DLLs and source files developed in Borland Delphi which are needed for the Requester Replacer exercise described in Chapter 8, Requester Replacer Examples.

Mattypes.pas	Include file used by Regrp16.dpr and Reqrp32.dpr.
Reqrp16.dll	16-bit dynamic link library.
Reqrp16.dpr	16-bit Delphi project file.
Reqrp16.res	16-bit Delphi resource file.
Reqrp16.txt	Textfile containing the filename of the 16-bit dynamic link library and the procedure name.

Reqrp32.dll	32-bit dynamic link library.
Reqrp32.dpr	32-bit Delphi project file.
Reqrp32.res	32-bit Delphi resource file.
Reqrp32.txt	Textfile containing the filename of the 32-bit dynamic link library and the procedure name.
Reqmain.dfm	Delphi form definition file.

The Reqrepl\Msvc Subfolder

This subfolder contains the DLLs and source files developed in Microsoft Visual C++ which are needed for the Requester Replacer exercise described in Chapter 8, Requester Replacer Examples.

Reqrp32.dll	32-bit dynamic link library.
Reqrp32.mak	Microsoft Visual C make file.
Reqrpl.c	Microsoft Visual C source file.
Reqrpl.def	Microsoft Visual C definition file.
Reqrp.h	Microsoft Visual C header file.
Resource.h	Resource interface file.

Chapter 7:
Using Requester Replacer

This chapter describes how to set up a Matterhorn session to use the Requester Replacer feature from the point when you load the relevant requester into Screen Designer until the user clicks a button associated with a Requester Replace. Issues related to programming the DLLs for a Requester Replace are presented and detailed.

The chapter is organized as follows:

- Using Requester Replacer
- Programming the DLL
- Defining a Requester Replace
- Testing the Requester Replace

Using Requester Replacer

This chapter provides guidance for using Requester Replacer and describes how to prepare a DLL for a Requester Replace. As mentioned in the previous chapter you need two things in order to replace a requester with a DLL:

1. A DLL with an appropriate interface.
2. A Requester Replace definition in Screen Designer.

On the following pages we will elaborate on these two issues.

Programming the DLL

On the following pages we assume that you understand what a DLL is and the way it is used.

There are two main issues concerning programming the DLL: getting the interface procedure right and addressing the parameters. When programming a DLL which is to be used in a Requester Replace, it may prove useful to start with one of the sample demos located in the \Demos\Reqrepl subfolder.

The interface procedure is a ordinary exported procedure. It does not return a value (in C syntax it is a `void` procedure). Matterhorn assumes that the procedure uses the *stdcall* calling model. In Visual C 2.0 this is denoted by the `WINAPI` function modifier. A valid interface procedure prototype may look like this:

```
void WINAPI ReplacedRequester (TArguments
       *arguments);
```

The *arguments* parameter is a pointer to a TArguments structure. In C syntax the structure is defined as:

```
typedef struct TArgumentsStruct
{
  char          requester_name[32];
  pCallBack     *callback_proc;
  short         param_count;
  TParam        param_list[100];
} TArguments;
```

requester_name identifies the called requester which have been replaced. It is a null-terminated string. Using this identifier you can differentiate between replaced requesters that use the same DLL and interface procedure.

The CallBack function is used to provide the DLL with information about the Matterhorn session. For instance, the DLL can obtain the Session Name and differentiate among sessions. For more information on the CallBack function, see section *CallBack Functionality of Requester Replacer.*

param_count determines how many arguments are transferred from the calling requester to the replaced requester (linkage section). The *param_list* is an array of structures that describes the transferred parameters. In C syntax the parameter structure is defined as:

```
typedef struct TParamStruct
{
  short len;
  void *addr;
} TParam;
```

The *len* variable determines the size of the parameter while *addr* is a pointer to the parameter. Notice that the format of the

parameter is defined by the Scobol program. You cannot change the size of a parameter.

In order to access a parameter you can use a statement like this:

```
strcpy(s, arguments->param_list[1].addr)
```

Assuming that the second parameter is a null-terminated string, the above statement copies the string to a variable *s*.

CallBack Functionality of Requester Replacer

The Callback procedure that is transferred in a call to a Requester Replace DLL provides you with information about the calling Matterhorn session. The procedure is declared as (C syntax):

```
void (WINAPI pCallBack)(unsigned short FuncId,
       unsigned short * ReturnCode, void *Data);
```

Notice that the function is using the *stdcall* calling model denoted in MSVC with the function modifier WINAPI.

The first parameter FuncId identifies what information to return in the *data* parameter:

Value	**Information**
1	Pathway name
2	System name
3	Session name
4	Terminal file

If the function succeeds ReturnCode is 0 (zero) upon return, otherwise non-zero.

The Data parameter points to an allocated memory space that contains the desired information upon return. All information is currently represented as null-terminated strings. If the function fails, the contents of the memory space is undefined.

Defining a Requester Replace

Start Screen Designer and load a requester into the main work area. Once the requester is loaded, the **Define** menu becomes accessible and you are ready to define a Requester Replace.

1. Select **Requester Replace** from the **Define** menu in Screen Designer. This opens the **Requester Replace** dialog box.

   Figure 8.3: *Enter the name of the requester you wish to be replaced by a DDL.*

2. In the **Requester Replace** dialog box, enter the name of the requester you wish to replace by a DLL, and click **Create**. This will take you to the **Create Requester Replace** dialog box (see Figure 8.4).

3. In the field **Library Filename**, enter the name of the DLL to address.

   You must make sure that Matterhorn is able to locate the DLL. One way to do this is to provide a valid path, such as “C:\Matthorn\Test.dll” (remember the file extension “DLL”). The DLL can also be located on a network drive.

4. In the field **Procedure Name**, enter the name of the procedure to call. The procedure name has to match an exported procedure in the DLL. Note that the string representing the procedure name is case sensitive. This means that you have to provide the exact case of each letter in the procedure name.

Figure 8.4 *The **Create Requester Replace** dialog box*.

There are a number of ways to check which procedures are exported. In Windows 95 and NT 4.0, for instance, you can use the Quick View facility of the Windows Explorer. You can also use utilities like Borland's TDump.

5. Click **OK** to return to the **Requester Replace** dialog box, and **OK** again to return to Screen Designer.

If you wish to base the Requester Replace on an existing definition, type the name of the requester in the **Base Requester Replace** on field or select it from the dropdown list. The DLL and procedure name of the definition associated with this requester, will be inserted in the fields **Library Filename** and **Procedure Name**.

Editing a Requester Replace

You can also edit existing Requester Replace definitions from Screen Designer:

1. Select **Requester Replace** from the **Define** menu in Screen Designer to open the **Requester Replace** dialog box.

2. In the **Requester Replace** dialog box, indicate the name of the requester whose Requester Replace definition you wish to you change, and click **Edit**.

3. In the **Edit Requester Replace** dialog box, change the contents of the fields **Library Filename** and **Procedure Name**.

4. Click **OK** to return to the **Requester Replace** dialog box, and **OK** again to return to Screen Designer.

Figure 8.5 *The **Edit Requester Replace** dialog box*.

Debugging the Requester Replace

To test the Requester Replace, run the Matterhorn session by clicking the relevant icon in your Matterhorn group folder. If you experience problems, open the **Messages** window. This window displays error messages that may assist you in debugging the requester replace.

Chapter 8:
Requester Replacer Examples

This chapter presents examples and exercises which are intended to show you how Requester Replacer may be used.

The chapter is organized as follows:

- Requester Replacer Demo
- Hotels

Requester Replacer Demo

As part of your MattOpen package you find the files needed for the Requester Replacer demo described in this section. The demo is intended to demonstrate the functionality of requester replacing. You must create the Requester Replacer definition for the demo yourself using Screen Designer. Here you indicate which requester you wish the DLL to replace. We enclose DLLs developed in both 16- and 32-bit Borland Delphi and Microsoft Visual C++. These are stored in the Demos\Reqrepl subfolder.

Creating the Requester Replace Definition

In the example below, MATTRQ04 is the requester to replace. The ReqRep32.dll is the DLL (developed in 32-bit Borland Delphi) that replaces MATTRQ04 and ReplacedRequester is the procedure to call in the DLL.

Figure 8.1: *The **Create Requester Replace** dialog box. In this example the Requester Replace is defined for MATTRQ04.*

Running the Requester

When the call to the requester is made, the window below is presented. The **Replaced Requester** window shows which requester has been called and the number of parameters (if any) that has been passed from the Matterhorn session. The values of the parameters are also shown.

Figure 8.2: *Screen of the Requester Replacer demo.*

In the **Replaced Requester** window you may click the **CallBack** button which retrieves various information about

the current Matterhorn session, including the Pathway system, the session name, the terminal file name, etc. The Callback function is described in Chapter 7, Using Requester Replacer.

Hotels

This example demonstrates how the Requester Replacer feature may be used to brush up a Tandem application and provide more functionality.

At a travel agency that services thousands of customers, the requester that shows information about hotels has been replaced by a DLL, which provides more elaborate information about the hotels.

When the customer dials in to order a tour and asks for further information about a specific hotel, the sales person clicks the **Hotels** button on the screen. The button activates the hotels.dll whose screen is depicted in the figure below.

Figure 8.3: *The hotels.dll provides detailed information about hotels.*

Note that this demo is not a part of your MattOpen package.

Creating the Requester Replace Definition

For this example, the following Requester Replace definition has been entered in the **Create Requester Replace** dialog box:

Figure 8.4: *The Create Requester Replace definition.*

Part 3

The Matterhorn Macro Language

Part 3 discusses the Matterhorn macro language which you may use to create and run macros from your legacy applications and/or client applications. Note that we will only present those statements of the macro language which require a MattOpen license. Also in this part, a series of examples may serve as inspiration when using the Matterhorn macro language.

Chapter 9:
The Matterhorn Macro Language

This chapter introduces the Matterhorn macro language, which you may use to create and run macros from your legacy applications and/or client applications. These macros may be used to set up DDE-conversations between your legacy requesters and any DDE-compatible application, call procedures in a Windows DLL, or a combination of the two.

The chapter is organized as follows:

- The Matterhorn Macro Language
- Enabling the Matterhorn Macro Language
- The DDE Statements
- The CallDLL Statement

The Matterhorn Macro Language

The Matterhorn macro language is an integral part of the Matterhorn Suite. The macro language enables you to create macros and integrate them with your Matterhorn sessions. You may use macros, for instance, to speed up and simplify administrative tasks or exchange data with other applications that support DDE or DLL. The Matterhorn macro language provides you with several related options:

- create macros and run them from your legacy Pathway applications in order to set up dynamic-data exchange (DDE) conversations with other applications that support DDE.
- create macros in the macro language of the client application to call your legacy Pathway applications.
- create and run macros that calls a Windows DLL.
- create macros that combine DLL and DDE technology.
- Matterhorn may operate both as DDE-client and DDE-server.

NOTE Some of the statements of the Matterhorn macro language require no MattOpen license. These statements, which are not related to the DDE- or DLL-technologies, are described in Chapter 7, in the Screen Designer Setup and Reference Guide.

Types of Macros

Using the Matterhorn macro language, you may create macros which operate in the following ways:

- Matterhorn operating as DDE-client, requesting data from any application with DDE-server capabilities, for instance Microsoft Word, Excel or Access, or from other Matterhorn sessions (see Figure 9.1). In this case, the entire macro is created in Screen Designer.

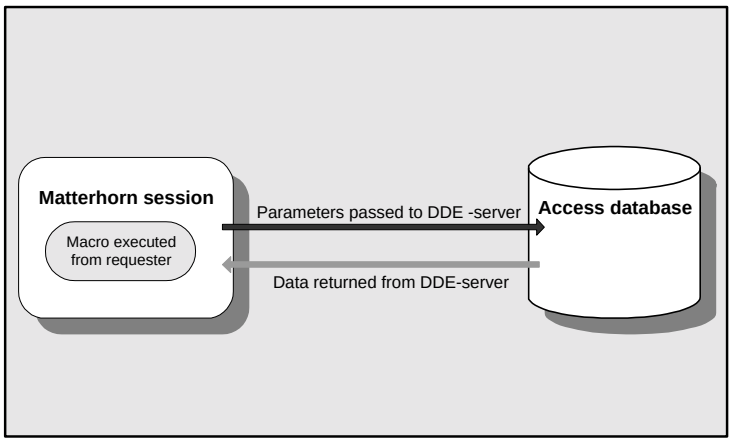


Figure 9.1: *Matterhorn operating as DDE-client, requesting data from Access.*

- Any DDE-compatible application operating as DDE-client, requesting data from one or several Matterhorn sessions or from other DDE-compatible applications (see Figure 9.2). In this case, the entire macro is created in the client application.

- Matterhorn and the DDE-compatible application both operating as client and server in turn, requesting data from each other or from other applications that may

operate as DDE-servers. For instance, a Screen Designer macro, which is run from a Matterhorn session, may activate a macro in Word, which in turn will request data from another Matterhorn session (see Figure 9.3).

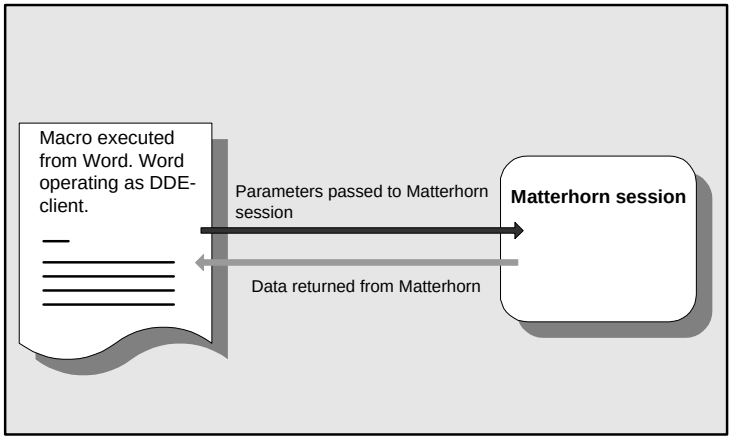


Figure 9.2: *Word operating as DDE-client, requesting data from a Matterhorn session.*

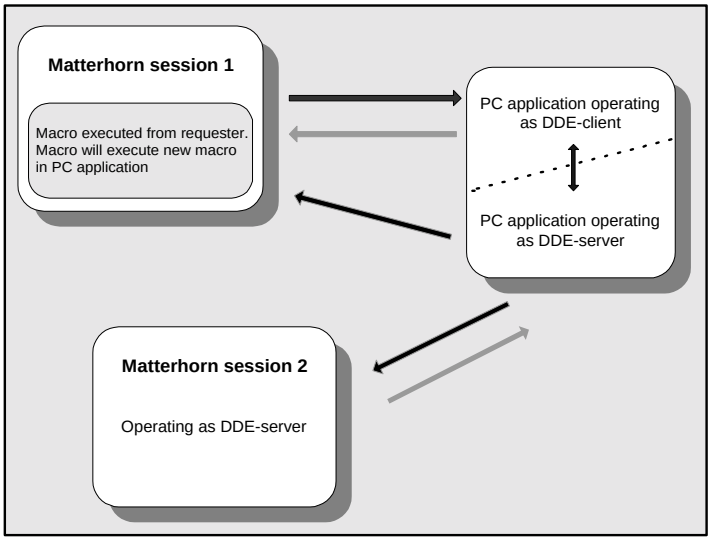


Figure 9.3: *Matterhorn macro executing a Word macro, which is requesting data from a second Matterhorn session.*

- Matterhorn operating as client making a call to a Windows DLL (see Figure 9.4).

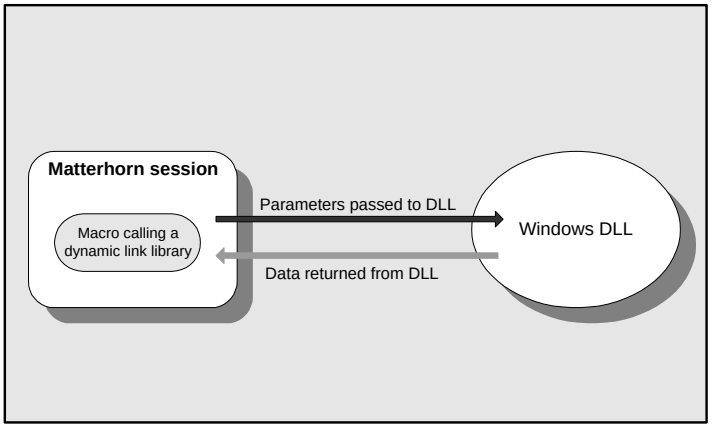


Figure 9.4: *A macro calls a Windows DLL from the requester.*

Enabling the Matterhorn Macro Language

The Matterhorn macro language is an integral part of Screen Designer, but you cannot use those statements which are used for DDE- and DLL-macros without a valid MattOpen license. The remaining statements are described in Chapter 7 in the Screen Designer Setup and Reference Guide.

To enable the statements used for DDE and DLL macros you merely have to transfer the new license file, LICENSE, from your MattOpen diskette to your Tandem. The next time you launch Screen Designer, you will be able to create macros containing the relevant DDE- and DLL statements.

The DDE Statements

The Matterhorn macro language provides the following statements for DDE conversation:

- DDEInitiate
- DDEExecute
- DDEPoke
- DDERequest
- DDETerminate
- DDETerminateAll

The statements are presented and detailed in Chapter 10, The DDE Statements.

The CallDLL Statement

The Matterhorn macro language provides the following statement to call a Windows DLL:

- CallDLL

The statement is presented and detailed in Chapter 11, The CallDLL Statement.

Combining Macros with Dynamic Interface

You may combine the MattOpen Dynamic Interface with macros. Imagine a situation where your system has already been set up to use the Dynamic Interface. Old and new has been successfully integrated. Now you wish to extend a legacy requester in the system, for instance with features for validating the contents of certain fields. In this case, you would write a macro in Screen Designer which makes the call to the DLL that validates the contents of the fields, and then associate the macro with a button or another screen object. At runtime you would make a call to the requester using the Dynamic Interface, and then have the contents of specific edit fields validated by activating the macro.

Chapter 10:
DDE Statements

This chapter presents the syntax and application of the DDE statements of the Matterhorn macro language which may be used to set up DDE-conversations between your legacy requesters and any DDE-compatible applications.

The chapter is organized as follows:

- The DDE Statements
- Parameter Types in DDE Statements
- DDEInitiate
- DDEExecute
- DDEPoke
- DDERequest
- DDETerminate
- DDETerminateAll
- DDE Support Files

The DDE Statements

Matterhorn macro language provides the following statements for DDE conversation:

- DDEInitiate
- DDEExecute
- DDEPoke
- DDERequest
- DDETerminate
- DDETerminateAll

Before you proceed with the descriptions of the statements, however, you should study the section *Parameter Types in DDE Statements.*

Parameter Types in DDE Statements

On the following pages we will describe the various DDE statements of the Matterhorn macro language. Each statement takes one or several of the following two types of parameters:

- Integer
- Identifier

The *Integer* parameter may be any whole number between 1 and 10.

The *Identifier* parameter may be any one of the four following types; *string*, *user variable*, *DSC variable*, and *edit field*.

The String Identifier

The string identifier is a string constant. String constants must be enclosed in quotation marks as in "*string*". If you should ever need to use quotation marks *within* a string you must use the ""*string*"" syntax.

Note that you may also use the concatenation operator '&' to build new strings, "Tom "&"Jones" = "Tom Jones".

The User Variable Identifier

You may use up till 255 user variables. User variables are internal storage and may be used as you please. For the user variable identifier you use the following notation; VAR[1].

The DSC Variable Identifier

The DSC variable identifier identifies a is a descriptor in working storage. For DSC variables you use the following notation: dsc[70].

The Edit Field Identifier

The edit field identifier identifies a specific edit field in the current requester. As you load a requester into Screen Designer, each edit field in the requester will have a unique object name assigned to it.

Rather than referring to edit fields by screen coordinates, Matterhorn refers to a field by its unique object name, enabling

the macros to locate a field no matter where it is placed on the screen.

Note that only edit fields in requesters that have been designed with Screen Designer will be assigned an object name.

The syntax of this name is *Edit<number>*, as in Edit1, but you may change the object name. Note that the maximum length of an object name is 20 characters.

DDEInitiate

The DDEInitiate statement initiates a DDE-conversation with the server application and opens a DDE-channel through which the conversation may take place.

When you start a DDE-conversation using DDEInitiate(), you reserve a channel for a specific topic recognized by the server application. In Microsoft Word, for example, each open document is a separate topic.

The statement has the following syntax:

```
DDEInitiate(Channel:Integer,Application:Identifier,
        Topic:Identifier)
```

Channel The channel number used for the DDE-conversation.

Application The application's EXE-file. Note that the application should appear from your PATH-statement. If the Matterhorn Client is to operate as DDE-server, remember that the application name of Matterhorn is Mattwin.

Topic This parameter may either be a string identifying a document or a spreadsheet, for instance, or an edit field in the current Matterhorn session.

Many applications that support DDE, including Matterhorn, recognize a topic named System, which is always available and can be used to find out which other topics are available. The string topic associated with a Matterhorn session is a so-called session identifier. The session identifier is the current session name.

DDEInitiate Exemplified

In a Matterhorn session macro, the following statement appears:

```
DDEInitiate(1, “WINWORD”,”LETTER.DOC”)
```

When the user runs the macro from the Matterhorn session, this statement will reserve channel 1 for DDE-conversation with Word and the document LETTER.DOC. If Word is not currently running, the macro will ask you to confirm that the application is launched.

```
DDEInitiate(1, Edit1, Edit2)
```

To run the macro, the user must make entries for the fields Edit1 and Edit2 (an application name and a topic). The statement will then reserve channel 1 for DDE-conversation with the specified application name and topic and launch the application.

DDEExecute

The DDEExecute statement sends a command or series of commands to an application through a dynamic-data exchange. Use this command, for instance, to launch a Word macro from a Matterhorn session macro.

The statement has the following syntax:

```
DDEExecute (Channel:Integer, Command:Identifier)
```

Channel The channel number of the DDE-conversation as provided by DDEInitiate().

Command A command or series of commands recognized by the server application, for instance a macro or a function key. You can also use the format described under SendKeys to send specific key sequences. If the server application can't perform the specified command, an error occurs.

DDEExecute Exemplified

In many applications that support dynamic-data exchange, *Command* should be one or more statements or functions in the application's macro language. For example, in Microsoft Excel the XLM macro instruction to create a new worksheet is NEW(1). To send the same command through a DDE channel, you use the following instruction:

```
DDEExecute (1, "[NEW(1)]")
```

Note that some applications, including Matterhorn, require that each command (like macros) received through a DDE-

channel be enclosed in brackets. Keys and function keys must be enclosed in braces:

```
DDEExecute(1,”{F1}”)
```

You can use a single DDEExecute instruction to send more than one command. For example, the following instruction tells Microsoft Excel to open and then close a worksheet:

```
DDEExecute (1, "[NEW(1)][FILE.CLOSE(0)]")
```

Note that there is no space between commands in brackets; a space character between the commands would cause an error. The preceding instruction is equivalent to the following two instructions:

```
DDEExecute (1, "[NEW(1)]")
DDEExecute (1, "[FILE.CLOSE(0)]")
```

Another example: In a Matterhorn session macro, the following statement appears:

```
DDEExecute (1, Edit3)
```

To run the macro, the user must make an entry for the field Edit3 (the command name) prior to launching the macro.

A third example: In a Word macro, the following statement appears:

```
DDEExecute(1,”[SetCursor(Edit5)]{Return})
```

The Word macro is designed to initiate a Matterhorn session for DDE-conversation with Word. The SetCursor command will then be executed, placing the cursor in the edit field

identified by Edit5. Subsequently, the edit field will receive a Return keystroke. (The SetCursor statement is described in the Screen Designer Setup and Reference Guide).

DDEPoke

The DDEPoke statement uses an open DDE-channel to send data to an application. When you start a DDE-conversation using DDEInitiate(), you open a channel to a specific topic recognized by the server application. In Microsoft Word, for instance, each open document is a separate topic.

When you send information to a topic in the server application, you must specify the item in the topic you want to send information to. In Microsoft Excel, for instance, cells are valid items and are referred to using either the "R1C1" format or named references. DDEPoke sends data as a text string; you cannot send text in any other format, nor can you send graphics.

The statement has the following syntax:

```
DDEPoke (Channel:Integer, Item:Identifier,
        Data:Identifier)
```

Channel The channel number of the DDE-conversation, as provided by DDEInitiate().

Item An item within a DDE-topic. If the server application does not recognize *Item*, an error occurs. In Word, an item may be a bookmark field.

Data The data to be sent to the server application.

DDEPoke Exemplified

In a Matterhorn session macro, the following statement appears:

```
DDEPoke(1, “NAME”, Edit4)
```

The statement will use channel 1, which has previously been reserved for a DDE-conversation with Word and the document LETTER.DOC. The contents of Edit4 in the Matterhorn session will then be transferred to the bookmark field NAME in the Word document.

DDERequest

The DDERequest statement uses a DDE-channel to request an item of information from a server application and transfer it to the client application. When you start a DDE-conversation using DDEInitiate(), you reserve a channel for a specific topic recognized by the server application. In Microsoft Word, for instance, each open document is a separate topic.

When you request information from the topic in the server application, you must specify the item in the topic whose contents you are requesting. In Microsoft Excel, for example, cells are valid items and they are referred to using either the "R1C1" format or named references.

DDERequest() returns data as a text string only; if the function is unsuccessful, it returns an empty string (""). Text in any other format cannot be transferred, nor can graphics.

The statement has the following syntax:

```
DDERequest (Channel:Integer, Field
        Name:Identifier, Item:Identifier)
```

Channel The channel number of the DDE-conversation as provided by DDEInitiate().

Field Name The edit field in the Matterhorn session, where the requested data will be inserted. The edit field may also be identified by a descriptor from working storage or a user variable.

Item The item within a DDE-topic recognized by the server application. DDERequest() returns the entire contents of the specified item. If the server application doesn't recognize *Item*, an error occurs.

DDERequest Exemplified

In a Matterhorn session macro, the following statement appears:

```
DDERequest (1, Edit5, "NAME")
```

When the macro is executed, the statement will use the open DDE-channel to request the content of the item "NAME" and insert it in the edit field identified by Edit5 in the Matterhorn session.

Matterhorn and other applications that support DDE recognize a topic named System. Three standard items in the System topic are described below. Note that you can get a list of the other items in the System topic using the item SysItems.

SysItems
Returns a list of all items in the System topic.

Topics
Returns a list of available topics.

Formats
Returns a list of all the Clipboard formats supported by the server application.

DDETerminate

The DDETerminate statement closes the specified dynamic-data exchange (DDE) channel. To free system resources, you should close channels you are not using.

The statement has the following syntax:

```
DDETerminate(Channel:Integer)
```

Channel The channel number of the DDE-conversation as provided by DDEInitiate().

DDETerminate Exemplified

In a Matterhorn session macro, the following statement appears:

```
DDETerminate(1)
```

The statement closes the DDE-conversation that has previously been reserved by channel 1.

DDETerminateAll()

The DDE TerminateAll statement closes all dynamic-data exchange (DDE) channels opened by Matterhorn. Using this statement is the same as using a DDETerminate statement for each open channel. DDETerminateAll does not cause an error if no-DDE channels are open.

If you interrupt a macro that opens a DDE-channel, you may inadvertently leave a channel open. Open channels are not closed automatically when a macro ends, and each open channel uses system resources.

The statement has the following syntax:

```
DDETerminate()
```

DDE Support Files

As part of your MattOpen package you find a couple of sample macros. These are stored in the Demos\Dde subfolder.

The Demos\Dde Subfolder

The Demos\Dde Subfolder contains two sample macro text files:

Mattdde.txt Text file containing the statements of a Matterhorn session macro that performs DDE with Word.

Excelldde.txt Text file containing the statements of an Excel macro that performs DDE with a Matterhorn session.

Chapter 11:
The CallDLL Statement

This chapter is intended to explain the syntax and application of the CallDLL statement, which may be used to call procedures in a Windows DLL.

The chapter is organized as follows:

- The CallDLL Statement
- Syntax of the CallDLL Statement
- Using the CallDLL Statement
- Programming the DLL
- Files of CallDLL

The CallDLL Statement

The CallDLL statement, as the name suggests, provides an additional feature to the Matterhorn macro language, namely the ability to call a DLL during runtime. The chapter is primarily aimed at programmers. We assume that you understand what a DLL is and the way it is used. In order to use the CallDLL statement you need to prepare two things:

1. A macro containing a CallDLL statement (see Chapter 12, Using Matterhorn Macro Language).
2. A DLL with an appropriate interface (read this chapter).

Syntax of the CallDLL Statement

The statement has the following syntax:

```
CallDLL(DLLName,DLLProcedure,Param1,…ParamN, Identifier)
```

DLLName Name of dynamic link library to be called. The DLL name may also be identified by a descriptor from working storage or a user variable.

DLLProcedure Name of procedure to be executed. The procedure name may also be identified by a descriptor from working storage or a user variable. The procedure to be called in the DLL is referred to as the *interface procedure*.

Parameters The edit field(s) in the Matterhorn session whose contents will be processed during the call. The fields may also be identified

by descriptors from working storage or user variables.

Identifier See the section Parameter Types in DDE Statements in Chapter 10.

Using the CallDLL Statement

The CallDLL statement takes several parameters. The first parameter identifies the DLL and the second, the interface procedure. The remaining parameters are transferred to the DLL as arguments. A valid CallDLL statement may look like this:

```
CallDLL("The_dll.dll", "ProcName", edit0, edit1);
```

You need to make sure that Matterhorn is able to locate the DLL. One way to do this is to specify the entire path of the DLL, such as "C:\Matthorn\Testdll.dll" in the CallDLL statement (remember to include the file extension, "dll"). The DLL can also be located on a network drive.

The procedure name has to match an exported procedure in the DLL. Note that the string representing the procedure name is case sensitive. This means that you have to provide the exact case of each letter in the procedure name.

There are a number of ways to check which procedures are exported. In Windows 95 and NT 4.0, for instance, you can use the Quick View facility of the Windows Explorer. You can also use utilities like Borland's TDump.

Programming the DLL

There are two main issues which are important when programming the DLL:

1. Getting the interface procedure right.
2. Addressing the arguments.

When programming a DLL for use with the CallDLL statement it may prove useful to check the sample demos, which are installed in the Demos\CallDLL folder. Matterhorn imposes no limits as to the functionality of a DLL.

The interface procedure is an ordinary exported procedure. It does not return a value (in C syntax it is a `void` procedure). Matterhorn assumes that the procedure uses the *stdcall* calling model. In Visual C 2.0 this is denoted by the `WINAPI` function modifier. A valid interface procedure prototype may look like this:

```
void WINAPI DLLCalled (short param_count,
       TArguments *arg);
```

The DLL is loaded and unloaded each time the CallDLL statement is called. This means that you cannot have any context information in the DLL.

The *param_count* parameter determines how many arguments are transferred to the interface function. The *arg* parameter is a pointer to a TArguments structure which is an array of string pointers. In C syntax the structure is defined as:

```
typedef struct TArgumentsStruct
{
  char *arg[255];
} TArguments;
```

Each element points to a null-terminated string and may contain as many as 255 characters. In order to deference an argument you can use a statement like this (C syntax):

```
strcpy(s, arg->arg[1]);
```

This will copy the second argument to the variable *s*.

Files of CallDLL

As part of your MattOpen package you find a series of files related to the CallDLL statement. These comprises header files and include files and a series of demos.

The Include\Delphi Subfolder

This folder stores an include file for Borland Delphi.

Calldll.pas Include file. The file defines the types needed for implementing DLLs for use with CallDLL.

The Include\Msvc Subfolder

This folder stores a header file for Microsoft Visual C++.

Calldll.h Header file. The file defines the types needed for implementing DLLs for use with CallDLL.

The Demos\Calldll Subfolder

The Demos\Calldll subfolder contains the files for a series of CallDLL demos. The demos are detailed in Chapter 12, Macro Examples.

The Delphi Subfolder

The Delphi subfolder contains the files needed for the 16- and 32-bit Borland Delphi CallDLL demos.

Call16.dll	16-bit dynamic link library.
Call16.dpr	16-bit Delphi project file.
Call16.res	16-bit Delphi resource file.
Call32.dll	32-bit dynamic link library.
Call132.dpr	32-bit Delphi project file.
Call32.res	32-bit Delphi resource file.
Macro16.txt	Contains the macro you need to call the Calldll16.dll.
Macro32.txt	Contains the macro you need to call the Calldll32.dll.

The Msvc Subfolder

The Msvc subfolder contains the files needed for the Microsoft Visual C++ CallDLL demo.

Call.dll	Dynamic link library.
Call.mak	Microsoft Visual C make file.
Calldll.c	Microsoft Visual C source file.
Calldll.def	Microsoft Visual C definition file.
Calldll.h	Microsoft Visual C header file.

Chapter 12:
Using Matterhorn Macro Language

This chapter describes the principles of creating and integrating macros in Screen Designer. The chapter is organized as follows:

- Creating Macros in Screen Designer
- Associating the Macro with an Object
- Testing the Macro

Creating Macros in Screen Designer

The Matterhorn macro language is an integral part of Screen Designer, which means that Screen Designer is the development tool you use when creating macros. It is also from Screen Designer that you associate the macros with the objects of a requester.

In this context, “associating” the macro with an object refers to the process of relating the macro to either a button, an edit field, a multimedia object, a picture or the requester object. The user then activates the macro from the requester by clicking a button, entering certain data in an edit field, clicking a picture or a multimedia object, etc.

There are no formal limits as to the number of macros you may create for and activate from the same requester. For instance, you may create different macros for the same requester in order to retrieve different sets of values from other programs. The only limit is disk space.

Creating a Macro

Start Screen Designer and load a requester into the main work area. Once a requester is loaded into Screen Designer, the **Define** menu becomes accessible and you are ready to create the macro:

1. Select **Define**, **Macro** to open the **Macro** dialog box (see Figure 12.3).
2. In the **Macro Name** field, enter a name for the new macro and click **Create** to open the **Create Macro** dialog box (see Figure 12.4).

In the **Macro Description** field, enter the relevant macro statements (See Chapter 10, The DDE Statements and Chapter 11, The CallDLL Statement).

3. Click the **OK** button.

Figure 12.3: *The Macro dialog box.*

Figure 12.4: *Enter the macro statements in the* ***Macro Definition*** *field.*

Associating the Macro with an Object

In Screen Designer you may associate a macro with the following types of objects:

Buttons	Click the button to launch the macro.
Edit fields	Double-click the edit field to launch the macro.
Pictures	Click the picture to launch the macro.
Multimedia objects	Click the multimedia object to launch the macro.
Requester object	The macro will be executed as the user performs an accept of the screen, for instance when exiting or updating the requester or moving on to another requester.

Associating the Macro with an Object

The macro may be associated with the object when creating or formatting the object. To insert a new object and associate it with a macro:

1. From the **Object** menu, select the relevant **Insert** command, or click the corresponding button on the tool bar.

2. A blank object will be inserted at the centre of the main work area. Use the associated Object Inspector to format the object (see Figure 12.5).

3. If the object already exists, select the object with the mouse.

Figure 12.5: *Associate the macro with the object from the Object Inspector.*

4. Click the small arrow to the right of the **Macro** property name to display a list of available macros. Scroll down the list until you've located the desired macro.

5. Return to Screen Designer.

Associating the Macro with the Requester Object

If you associate the macro with the requester object, you can have the requester launch the macro as the user performs an accept of the screen, for instance when exiting or updating the requester or moving on to another requester.

To associate the macro with the requester object:

1. Click anywhere in the background of the requester to open the Object Inspector of the requester object.

2. Click the small button to the right of the **OnAccept** property name and locate the macro on the list.

3. Return to Screen Designer.

Figure 12.6: *Associate the macro with the requester object from the Object Inspector.*

Debugging the Macro

To test the macro, run the Matterhorn session by clicking the relevant icon in your Matterhorn group folder. Locate the relevant object and launch the macro.

If you experience problems, open the **Messages** window. This window displays error messages that may assist you in debugging the macro.

DDE to Matterhorn

Matterhorn can also operate as a DDE server. In this way an application can communicate with your legacy applications. For instance, MS Word can extract customer information from a database located on the Tandem by issuing a few DDE statements.

```
Cnv = DDEInitiate("mattwin", "mattwin")
Data$ = DDERequest$(Cnv, "Edit6")
DDETerminate(Cnv)
```

The above code retrieves the value of the edit field named “Edit6” from the legacy application into a local Word variable named Data.

Notice that the topic of the DDERequest is the name of the Edit field (the object name). This means that you can only fetch values one at a time. This also applies to DDEPoke statements.

If you want to have a Matterhorn session call another application using a macro and this application should use DDE to communicate with the same Matterhorn session, make

sure that the application specifies the string “DDELoopBack” as topic in its DDE connection statement. Otherwise Matterhorn will generate the error “DDE client rejected while running macro”.

Chapter 13: Macro Examples

This chapter is a collection of simple examples that is intended to show you how to create macros using the DDE statements and CallDLL statement of the Matterhorn macro language in Screen Designer. Simple as they are, the examples will serve as inspiration in your work with the Matterhorn macro language.

The chapter is organized as follows:

- Validating Country Codes
- Merging Letters in Word
- Launching a Macro from Excel

Validating Country Codes

In the first example, we will create a macro that validates the country code used for dialing to different countries in Europe. The macro is based on the CallDLL statement. When the user enters the name of the country in the field **Country** and double-clicks the field, it is automatically updated with the correct code used for telephone calls to that country.

Validation is performed by the procedure MapCountryCode in the DLL called Call.dll. The C source code depicted below is also stored in the Demos\Calldll\Msvc subfolder:

```
void WINAPI MapCountryCode (short param_count,
       TArguments *arg)
{
  char *country_code;

  // Fetch first parameter
  country_code = arg->arg[0];

  // Uppercase code
  strupr(country_code);

  if (!strcmp(country_code, "F"))      // France
    strcpy(arg->arg[0], "0033");
  else if (!strcmp(country_code, "GB"))// Gr.Britain
    strcpy(arg->arg[0], "0044");
  else if (!strcmp(country_code, "D")) // Germany
    strcpy(arg->arg[0], "0049");
  else if (!strcmp(country_code, "S")) // Sweden
    strcpy(arg->arg[0], "0046");
  else if (!strcmp(country_code, "I")) // Italy
    strcpy(arg->arg[0], "0039");
  else if (!strcmp(country_code, "E")) // Spain
    strcpy(arg->arg[0], "0034");
```

```
else if (!strcmp(country_code, "A")) // Austria
    strcpy(arg->arg[0], "0043");
  else if (!strcmp(country_code, "DK"))// Denmark
    strcpy(arg->arg[0], "0045");
  else if (!strcmp(country_code, "N")) // Norway
    strcpy(arg->arg[0], "0047");
  else if (!strcmp(country_code, "NL"))// Holland
    strcpy(arg->arg[0], "0031");
}
```

In order to call this DLL you must create a macro in Screen Designer.

Start Screen Designer and load the relevant requester into Screen Designer. Select **Macro** from the **Define** menu. In the **Create Macro** dialog box, enter the following statement:

```
CallDLL("Call.dll","MapCountryCode",Country)
```

In the macro the edit field Country contains and receives the country code.

When you have created the macro, associate it with the **Country** field using the Object Inspector. Remember to save the requester.

Merging Letters in Word

This Matterhorn session macro merges data from a customer file with a standard letter in Word and then initiates a Word macro that prints the letter. Its purpose is to print an order confirmation targeted at the person who has ordered a specific product. In the requester, the macro is associated with the button **Print Order Confirmation**.

The macro contains the following statements which are entered in the **Create Macro** dialog box:

```
DDEInitiate(1, "WINWORD", "dalmerge.doc")
DDEPoke(1, "Name", Edit2)
DDEPoke(1, "Street", Edit3)
DDEPoke(1, "PostCode", PostCode)
DDEPoke(1, "City", Edit5)
DDEPoke(1, "Country", Edit6)
DDEPoke(1, "ZipCode", Edit7)
DDEPoke(1, "Contact1", Contact1)
DDEPoke(1, "Product", Product)
DDEExecute(1, "[FILEPRINT]")
DDETerminate(1)
```

Technically, when the user clicks the button, this is what happens:

The macro initiates a Word session and opens the standard letter Dalmerge.doc. It then inserts the contents of the Tandem fields Name, Street, PostCode, ZipCode, Country, Contact1, Product, and Sales Person in the relevant places in the Word document (see Figure 13.1). The macro then executes the Word macro FILEPRINT, which sends the order confirmation to the printer.

Figure 13.1: *The standard letter, OrderConf.doc. The contents of the edit fields are inserted in the relevant places before the letter is printed.*

Launching a Macro from Word

In this macro Matterhorn operates as DDE-server. The Word macro retrieves all available session names in Matterhorn

Configuration and inserts them in a Word Document. Note that the macro is explained in the source code of the macro.

```
'
'The macro assumes that Matterhorn starts in the
       'MattRq01 requester.
'
Sub MAIN
'  Initialize DDE channel
  chnl = DDEInitiate("mattwin", "mattwin")

'  Type F3 to enter Session Setup
  DDEExecute chnl, "[DoFKey(" + Chr$(34) + "F3" +
       Chr$(34) + ")]"

' Type F4 (ReadNext) to get first requester
  DDEExecute chnl, "[DoFKey(" + Chr$(34) + "F5" +
       Chr$(34) + ")]"

'  When the same session name is returned twice
       'there are no more
  PrevSession$ = "NONEXSISTINGSESSION"
  Session$ = ""

  While Session$ <> PrevSession$
    PrevSession$ = Session$

'  Edit6 contains the session name
    Session$ = DDERequest$(chnl, "Edit6")

    If Session$ <> PrevSession$ Then
'  Insert the name at the end of the document
      EndOfDocument
         Insert Session$
      InsertBreak .Type = 6

'  Type F5 to get the next session
      DDEExecute chnl, "[DoFKey(" + Chr$(34) +
        "F5" + Chr$(34) + ")]"
```

```
    End If
  Wend

‘  Finish off by typing SF16
  DDEExecute chnl, "[DoFKey(" + Chr$(34) + "SF16"
       + Chr$(34) + ")]"

‘  Close DDE channel
  DDETerminate chnl
End Sub
```

Index

E

F

H

I

J

L

M