Norbert Kessel / Veit Wadewitz Gupta SQLWindows32 - An Introduction

Publishing House: www.GuptaBooks.com

Publishing House Dr. N. Kessel
Eifelweg 37
53424 Remagen
Tel./Fax: (0) 2228-493

eMail: webmaster@forstbuch.de

Translated from original German by Eva Tullier.

An additional book is available from the publisher, entitled "SQLBase – An Introduction". Further information and downloads are found at the publisher's homepage, above.

Publisher's homepage:	www.GuptaBooks.com (international)
	www.ForstBuch.de (Germany)
Gupta's homepage:	www.guptaworldwide.com
Printed by:	www.business-copy.com

Norbert Kessel / Veit Wadewitz

Gupta SQLWindows32
Introductory Manual to Programming Database Applications

ISBN: 3-935638-31-0

Contents

Foreword to the Second, Modified, and Expanded Edition

This book is intended to explain how to use the program packet Gupta SQLWindows to develop databases which are executable under Windows 32-bit operating systems. It is an introductory book of programming and as such, targeted toward beginners; due to its structuring and many examples, however, it may also prove helpful to advanced programmers.

Great emphasis has been laid upon presenting examples individually and completely. In this way it is hoped that the examples can easily be tested in the reader's own, smaller programs. It is, however, impossible to represent every functionality. Selection of topics is inherently subjective; nonetheless, the chosen selection should serve as a motivating start to programming database applications with this powerful tool.

This edition details Gupta Team Developer versions 2.1 and 3.0 (beta 4 at time of printing). The functionalities which were not or differently implemented in earlier releases (1.x) are indicated on their respective pages in this book.

Because the basic functionality of Gupta Team Developer has not changed significantly since the last version, the introductory capitals 1 and 2, with exception of the new section 1.2.3 An Excursion in Object Oriented Programming, have been included unchanged from earlier editions.

New additions are sections on the Component Development Kit, tooltip pushbuttons, and WebDeveloper, which were also already available in CTD 1.x.

Further topics include OLE DB database integration, ActiveX integration – with particular attention to Excel and Word –,COM server and client creation, and the Web XML table, which are available from Gupta beginning with Team Developer version 2.x. Especially these functionalities enable development of modern, performant applications with SQLWindows.

The examples of configuration given in this book can be downloaded from the publisher's homepage (www.guptabooks.com) under "Download".

We would like to thank the Gupta Company, with special thanks to Martin Teetz. Special thanks to EvaSara Toullier who translated into English.

Remagen and Lauf, January 2003

Dr. Norbert Kessel
Dr. Veit Wadewitz

Chapter 1
Introduction

A Few Comments About Hard Drive Directories

Complete installation of Gupta under Windows 32-bit creates several directories whose contents are briefly defined in the following.

The following directories are created on the hard drive selected for installation:

- **\Gupta (starting version 3.0) or \Centura** Includes the data necessary for programming database applications.
- **\Sql80:** If specified during installation that the local SQLBase be installed, this will include the data necessary to do so (it is alternatively possible to work with a database in a network directory; the above directory is then unnecessary).

The following table includes the subdirectories included in both directories, and the extensions of their files.

Directory	Subdirectory	File Type (Extension)
\Gupta 2.1	\AXLibs	ActiveX APLs
	\Deploy	Exe, Wse, Ini
	\Docs	HTM
	\Inc	H
	\Island	Dbs, Log
	\Samples	Apl, App, Bmp, C, Csv, Def, Dll, Exe, Ico, Ifx, Ini, Mak, Ora, Qrp, Reg, Sql, Wts, Xls, Cqt, Cpp, Dsp, Dsw, H
	\Sysproc	Sql
	\Templates	App
	\Tools	Apt, exe
	\Tutorial	App, Dll, Exe, Tlb, Ico, Gif
	\uninstall	Exe
SqlBase 8.0	\Alarms	Fil
	\Books	Pdf, HTML
	\include	Inl, H
	\island	Dbs, Log
	\Java	Class, HTM
	\lib	Lib, Dll
	\Redist	Setup.exe
	\relnotes_files	Release notes (Htm, Jpg)
	\samples	Sql, C, Dll, Java, Dsp, Dsw, Cpp, H, Rc, Mak, Exe, Vbp
	\scripts	Sql
	\src	Cpp, Dsp, Dsw, H, Rc, Rc2
	\Topic	HTM

The Directory DEPLOY

Gupta installation creates a directory named DEPLOY. This directory contains files with the extension WSE, INI, and an EXE file (deploy21.exe under SQLWindows 2.1).

The files create the environment necessary for running an application on a computer. In other words, after an application program has been written with Gupta, this application is saved as an EXE file (in Gupta over the menu option Project/Build Settings – Exe). Besides this EXE file and possibly the DBS file with the database – as well as ISO or BMP files, the customer must also be given all of the files included in the DEPLOY directory. Altogether, then, the following packet must be included in delivery of a program which has been developed with Team Developer:

- EXE file created with Gupta Team Developer;
- All files in the DELPOY directory;
- DBS file(s) including data;
- SQLBase (e.g., SQLBase Runtime or other SQLBase which customer obtains).

Tip: As mentioned above, all applications that will be installed on an external computer require the files included in the DEPLOY directory. There are two ways to do this: The CD with the complete development environment can be taken to the customer site and the files installed with the installation program. Otherwise, it is best to burn the files (in the current version these are 3 files with a total of abut 17 MB) from the DEPLOY directory onto a CD together with the application.

Other Important Programs for Installation

Besides the development environment, a few other programs are installed during Gupta installation. The following overview offers a list of the applications, their program names, and a brief description of each.

Name	File Name	Description
Gupta SQLWindows 2.1/3.0	Cbi21/30.Exe	Development environment
Repository Setup Wizard 2.1/3.0	Tmdbi21/30.Exe	Installs database for repository (used by Team Development)
Sample Application Explorer	Sampler.Exe	Sample application
SQLBase Database Engine 32bit	Dbnt1sv.exe	Database engine
SQLConsole DBA Utility 7.6/8.0	Sqlcon76/80.Exe	Program for database file maintenance
SQLTalk Interactive SQL	Sqltalk.Exe	Dialog program for SQL commands
Team Developer Help 2.1/3.0	Centura/Gupta.Hlp	Help file for development

		environment
Team Object Manager 2.1/3.0	Tmi21/30.Exe	Together with repository database for management of projects and their versions

The term “Gupta SQLWindows” is used in the literature and online documentation together with the term “Gupta Team Developer”, also in this book.

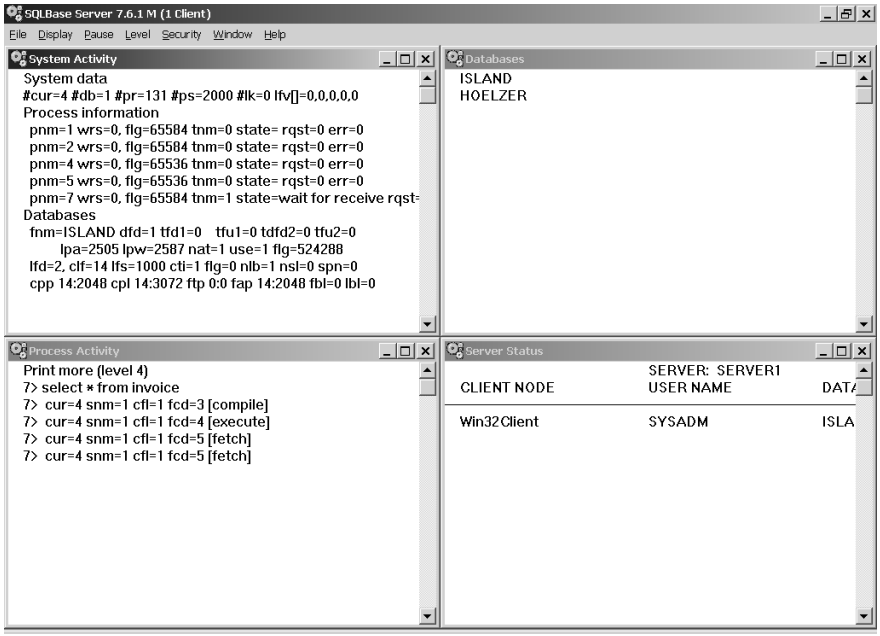
Some of the above programs and files are detailed below.

SQLBase 32 Bit Server

Normally a self-created application contains a connection to one or more databases. There is, then, usually a system which consists of two programs: the actual application itself, which was created by the programmer, and the database engine, which publishes the data that is saved in the DBS database. SQL commands, which are included in the application, are sent to the database engine and are there translated and valuated; result sets or messages may then be “sent” back to the application.

SQLBase Bit Server is the database engine that provides data, receives and executes incoming SQL commands, and sends result sets back to the application. If the function **SqlConnect()** is used in Team Developer, then, where applicable, the database is started (or the required router or ODBC driver); otherwise, a connection to the database is established (SQL handles, explained below, are required for this).

After the database engine has been started, clicking the icon in the status bar display the following image:



Illus.: Database engine with various windows with information (windows opened from the menu Display/All)

The individual windows offer information about currently opened databases and system load.

Files Which (Could) Appear When Working With Gupta Team Developer

Basically, working with Team Developer can create three types of files: program files, libraries, or COM servers.

- Program files can be saved in various formats: as **APP** file (in normal or compiled form), or as **APT** file (as normal or indented text).
- Library files contain functions which were written with Team Developer and saved in a separate file to make them utile for other applications. A command that integrates this APL or APD file is enough to make these functions utile for other programs. A description of how this functions is given below.
- COM servers are created in compiled form as **EXE** or **DLL** (local) or as MTS-DLL (**M**icrosoft **T**ransaction **S**erver).

Preparatory Literature

Besides this book, which provides a complete introduction to programming database applications, Gupta includes a few online books and documents which should definitely be read.

The following is an overview of the most important documents.

Title	Type	Where to Find	Contents
Introducing Gupta	File	Books Online	Introduction to Gupta
Developing with Gupta SQLWindows	File	Books Online	General tips about developing applications
Reporting with Gupta SQLWindows	File	Books Online	Programming reports
SQLConsole Guide	File	Books Online	Introduction to SQLConsole
SQLBase SQLTalk Command Reference	File	Books Online	Introduction to SQLTalk
Developing: Chapter 7 - SAL (Scalable Application Language)	File	Books Online (spyglass button)	SAL functions
Developing: Chapter 17 - Libraries	File	Books Online	Libraries
Extending: Chapter 11 - Visual Toolchest	File	Books Online	Advanced functions (see also VTTEST.APP in Samples directory)

How This Book is Composed

This book is divided into the following chapters:

Chapter 1 is an introduction to programming and offers tips to using the language SQL (Structured Query Language) and SAL (Scalable Application Language) as well as detailed descriptions of the objects usable in an application. Besides many examples of using Table Windows, here are extensive comments about printing data of every type.

Chapter 2 introduces a sample application which can be downloaded for free.

Chapter 3 details additional aspects of programming; e.g., tips about DDE, working in the net, using other databases, about barcodes, tooltips, and the CDK.

Chapter 4 treats ActiveX integration and creation of COM servers and clients.

Chapter 5 describes the development of web applications with SQLWindows.

Chapter 6 contains overviews and summaries of functions and messages.

Chapter 7 is this book's index.

Important Terms

Many terms are so fundamental that it is worthwhile to explain them briefly here. Extensive, detailed definitions are found in later sections.

Front-end, back-end, client-server: An application is termed front end which exchanges commands or data with a back-end (below) application; together, both are called a client-server system. Here, a front end is understood an application which was created with Gupta Team Developer; the term should, though, be viewed more globally, because basically (and basically means that there are exceptions) every Windows application can function as client or as server. For instance, WinWord or Excel (specifically, a document opened therein) can be used as client for data taken from a database, which was opened using SQLBase.

Database or **database engine**: A program that manages data which is saved in a DBS file. Specifically, new data records are added to a table by the databank engine; data records marked for deletion are deleted by the engine; result sets created by inquiries are created by the databank engine and sent to the front end from which the command was sent. In addition to these tables, databases include indexes and a few other objects which are usually created/deleted with the program SQLConsole (although it is also possible to create a databank within the Gupta Team Developer development environment: menu option Database/ Database Explorer).

Table: Data such as, for example, customer information, is saved in the form of a so-called table. Each table has a unique name and is defined by a structure. Tables are created, edited, or deleted by the user with the program SQLConsole or SQLTalk; they can include up to 2^{15} data records (Note: it is also possible to create, etc., data records within Team Developer; however, this should only be practiced with test data because it is better to use SQLConsole.). The term table is used in two forms; first, for tables that contain data which is saved in a database; and second, for so-called table objects, which are embedded in an application and there display e.g. data from a database table. Aside from tables which are created by a user, a database always contains such tables which are important to system operation; e.g., a table Syscolumns, in which all columns (of all tables) are listed. These tables can be used if an overview of the tables or table structures can be printed.

Index: An index is a help table which enables faster data selection, search, or assortment in a database. The response time is usually drastically minimized when selecting a few different data records from many, so that indexes are generally recommendable, even if data entry or deletion is somewhat slower.

Query: A Query is the command to select from a given set of data records a portion which meets a specification; e.g., to select from a group of invoices those which are yet unpaid.

Data type: Data types define what type of information is to be saved in a variable or in the column of a table. It is important to remember that file types must be defined during coding and in a table's structure (occasionally data types must be converted before they can be saved; this is explained below).

Normalization: Normalization removes redundancies; i.e., data is saved in such a way that, for example, a customer data record only appears once and all invoices which were created for this customer can be assigned to this customer using the customer number.

SQL (Structured Query Language): SQL is a language which can be used to create, edit, or delete databases, tables, data records, etc. This language is typically used to edit data records.

SAL (Scalable Application Language): SAL is the language used in Gupta Team Developer to code actions which should occur with objects on the screen. For example, a function named SalTblPopulate() is used to fill a table object in an application with the data records of a database table. Gupta Team Developer has over 400 such functions.

Chapter 1.1
Tips for Programming Database Applications

General

A database application is a program (an EXE file) that was created with Gupta Team Developer and includes objects (e.g., a table object, data fields, etc.). To display data from a databank table, SQL commands must be sent in the SAL language to the databank engine, which then valuates the data.

Applied Data

In this book, the introduction into programming uses data which has been transmitted to the hard drive during installation. It is found in the database named **Island.Dbs** (\Sqlbase\Island\Island.dbs). This should ensure a quick start. The following sections describe a few of the tables which are included in this database and used for this book (the appendix to this book includes an overview of all tables included in Island.Dbs).

SQL

Gupta uses two parallel languages, SAL (Scalable Application Lanuage) and SQL (Structured Query Language). The following offers a few fundamental comments about SQL.

The database language SQL is used for all access to a database. SAL, on the other hand, is used for application process control (e.g., when a window opens, a report is printed, etc.). Sometimes elements of both languages are used in combination; namely then, when a SAL function is invoked which works with data from tables. Later, an example of this will be given using the function SalTabPopulate(), a SAL function which is used to fill a table with data records from a database, and whose parentheses must enclose a SQL command.

Here, only a few SQL commands are given which enable data records to be entered into a table, to be edited, and to be deleted. This should ensure that even the SQL layman can quickly start programming databank applications.

Some SQL Commands:

Command	**Actions**
select customername, surname, forename from customer	Read and display contents of table Customer of columns Customer name, Surname, and Forename.

select, surname, forename from customer where customer number = 10000	Read content of columns Surname and Forename from data record with customer number 10000.
delete from invoices where status = 2	Delete from table Invoices all data records whose content in Status field is 2.
update item set price = price * 1.10	Replace in Item table content of Price field with the value of Price * 1.10; so, the prices of all items will be increased 10%.
select max from customernumber +1	Detect larges customer number and add the number 1 to it. Often used to create a new (numerical) customer number.

The commands can (and should) be tested with a front end program like SQLTalk, SQLConsole, or Database Explorer, for example, within Gupta.

Libraries

Libraries are files filled with functions. These functions (which can be written using SAL, or C) are often the results of complex applications in which certain actions are repeatedly necessary; e.g., opening a database, deleting data from forms, or reading data records from a table. Once the program code for these functions has been constructed so that they can be invoked using variables or parameters, the functions are saved in a separate APL file. This APL file does not include anything other than the functions – i.e., no forms or table windows. Implementing the APL file in an application (which typically ends with APP or APT) makes all functions available for this application. It is also possible to furnish the file Newapp.App (below) with this entry, so that the libraries are available in every new application. The source code for integrated libraries is displayed by default in blue in the main application and cannot be edited there.

Dynamic libraries (APD) are a second type of library. They include graphic components as well as functionality and are themselves “mini” applications. APDs are dynamically integrated in runtime and can communicate with the application which invokes them. In this way it is possible to share program windows between multiple applications without including the full source code in every application. APDs therefore mainly serve to conserve client resources. The APD’s functional code is transparent in the main application; the string !_Exported must be set in the APD after the declaration of top level objects and functions in order to see and target these objects in the main application.

Classes

A class is a template which specifies objects’ properties. For instance, a table class can be preset with variables so that later only the variables’ contents need to be defined. One of the first is the so-called standard class, in which objects’

properties (e.g., data fields) are preset. When a data field is entered in a form, the data type is preset as “string”; i.e., the data field just entered expects string data (numeric data, in contrast, cannot be represented and leads to an error message). Data fields in which numeric data must be represented are of course often required. In this case instead of each time changing the type of string to number and setting the alignment from left to right and setting the format (the number of decimal places), a new class is simply created, in which these properties are preset once. If thereafter a new data field is entered in a form, it is possible to choose between the standard classes; subsequent modification is then unnecessary.

Objects

An application’s object is a pushbutton, a table window, etc.; so, parts of the application which can be furnished with program code. Objects also process messages which can be processed when an event (like a mouse click) occurs. There are standard, quick, and self-defined objects, which are categorized within respective classes.

Generally there are two types of objects: **top level** or parent objects, which serve as containers for **child objects**. Top level objects can contain parameters, variables, functions, possibly menus and actions, and are changeable in size during runtime. Child objects can only contain actions. The only exception is the child table window, which will be discussed in more detail later.

Usually one starts out in Gupta using standard class objects to create smaller or even larger applications. At the latest, though, by the time a pushbutton is coded to minimize a form window or to close an application, the question arises whether there might not be a simpler way. There is; however, just when one is beginning, it’s complicated, because it is necessary to abstract and to recognize correlation (e.g., all pushbuttons to minimize a window).

As experience grows, though, it becomes apparent what is common (or, dissimilar) between similar modules. Correlation can then be so generally coded in the form of a class or an object, that it can be integrated into all future applications.

Messages

Messages are sent within SAL applications to provide information about specific program steps or performed actions. Thus the message “SAM_Click” offers a simply way to establish reference between a pushbutton and the user’s action (namely a click on this pushbutton). In addition to prefab messages, which begin with “SAM” (list found in appendix),it is possible to create and manage individual messages. These messages, which could begin with “AM” (can be anything but

should be distinguishable from SAM messages), make it possible to write more flexible programs.

This is of great import to programmers: programs under Windows no longer rigidly dictate what a user can or cannot do, as was possible and even normal with DOS programs. Now a program must be created so that it is possible to react anytime to every input from a user. The programmer must ensure in advance that all possible user input is correctly recorded and processed.

While this is not necessarily simple when using program languages like “C” or “Pascal”, Team Developer has already prepared for it; it already has the ability to react to multifaceted input. For example, the programmer only needs to check whether the user pushed the OK button in a previously active window. Because the program code which will then be executed is connected to this pushbutton, the user’s input can be further processed correctly.

Messages are triggered by events. These can come from hardware (mouse, keyboard), the operating system (e.g. timer), or the application (e.g., when a limit value is exceeded).

Gupta Team Developer distinguishes between three types of messages (more info is found in Team Developer under Help/Messages):

- Messages with the prefix SAM, which are predefined in the system,
- Messages with the prefix WM, which are predefined by Microsoft, and
- Individual, self-defined messages, which begin with, for example, AM (Am is not predefined but is normally used).

Examples of SAM Messages:

SAM Message	**Meaning/Trigger**
SAM_AppStartup	Starts application
SAM_App_Exit	Ends application
SAM_SqlError	Error in an SQL command
SAM_Click	Left mouse click (corresponds to WM_LBUTTONDOWN)
SAM_Create	After an object has been created
SAM_SetFocus	Object contains focus
SAM_Destroy	Object is destroyed (removed from screen)
SAM_Report*	Multiple messages for printing reports
SAM_DDE*	Multiple messages for DDE protocol

The messages can not be used randomly. For example, in an applications section application actions (i.e., at the very top) only three messages can be used (SAM_AppStartup, SAM_App_Exit, SAM_SqlError). In contrast, SAM_Create can be used with almost every object; in other words, the message can be used and edited for a form window as well as for a pushbutton.

Note: If the Visual Toolchest Library is integrated in an application, additional messages are available; like VTM_RightClick, VTM_MiddleClick, and VTM_RightDoubleClick, etc. These messages are edited just like SAM* messages and radically increase application flexibility.

Examples of WM Messages:

Messages with the prefix WM come from Microsoft Windows. They are defined in the file Windows.H and can be viewed there (the file is included with the MS Windows SDK and MS C++).

SAM Message	Meaning/Trigger
WM_LBUTTONDOWN	Left mouse button was pressed
WM-LBUTTONUP	Left mouse button was released
WM-RBUTTONDOWN	Right mouse button was pressed
WM-DDE*	Multiple messages for DDE

Examples of AM Messages:

These messages are predefined by the programmer; a certain amount of experience is usually necessary to recognize their great usefulness. The following overview can therefore not be “complete” and should here serve only as stimulus.

AM Message	Meaning/Trigger
AM_Empl_Select	When the table is filled for displaying employee data
AM_Inv_Select	Ditto, for the Invoice table (among the tables included in the Island.Dbs database)

To declare such a message, the following entry must be made in the program’s Constants/User section:

```
Constants
        System
        User
                Number: AM_Empl_Select = SAM_User + 1
```

Messages are always of the type number, because internally, numbers are always used. In addition to setting this type, the message’s name must also be entered (always case sensitive). Then comes the system constant SAM_User. This stands for the last defined message in the system; the number 1 is added to receive a new, as yet unused number. In the same way, the next message would receive an available number with SAM_User +2. This method avoids the chore of searching for available numbers; furthermore, a number thus coded can also be used in new Team Developer program versions without modification (new versions commonly use more system constants than older ones).

Messages can be sent within an application with help from the function SalSendMsg(), SalSendMsgToChiildren(), SalSendClassMessage(), SalSendClassMessageNamed(). The following construction would then ensure that, for example, the message to fill a table window with table data is sent:

```
On SAM_Create
        Call SalSendMsg(tblEmployee, AM_Empl_select, 0, 0)
```

This means that after the table window has been created, the message AM_Empl_select is sent.

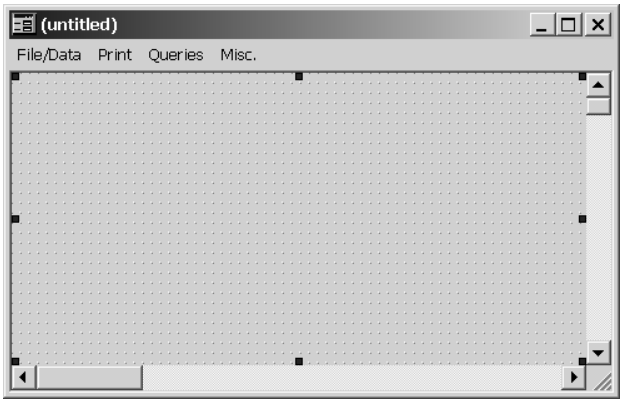
To receive this message elsewhere, the following code is possible:

```
On AM_Empl_Select
        Call SalTblPopulate(...)
```

User Control

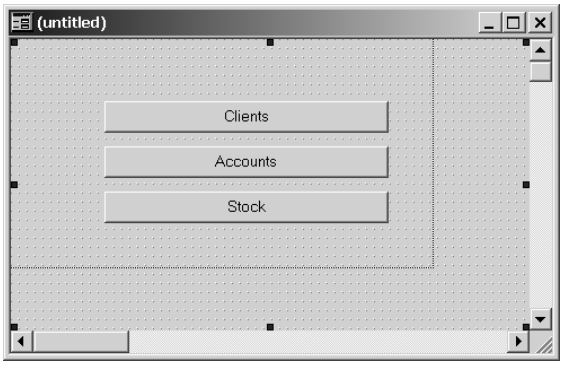
User control is understood here as the user's first contact with the application. This contact should be set up so that the user can work with the application without difficulty. Three basic options lend themselves for this:

- The application is controlled from a "classic menu".



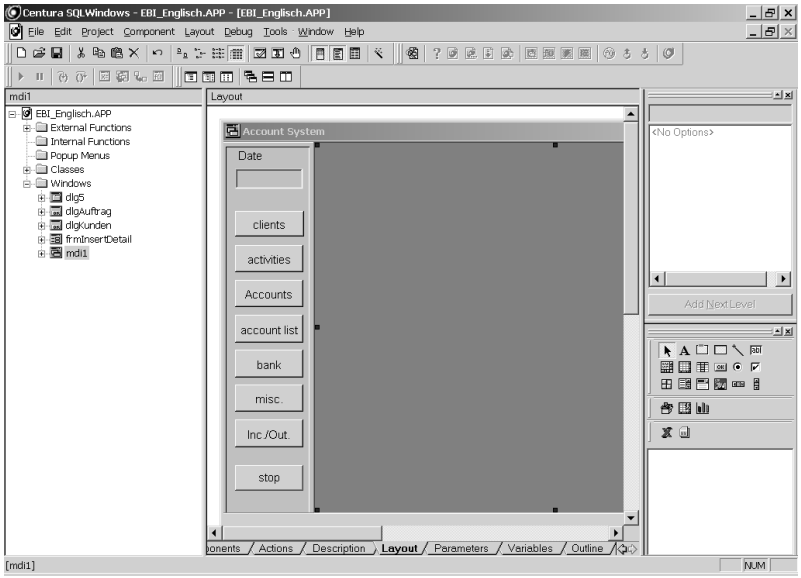
Illus.: Classic menu (displayed in a form window's layout mode)

- The application is controlled from a form window in which the individual program modules are invoked with pushbuttons.



Illus. Control from form window

- The application is controlled from a central MDI (Multiple Document Interface) window (see also sample application in Chap. 2).



Illus.: Central MDI window (the names of the child windows to be seen on the left)

A few comments to the possible and very sensible combinations between the three types:

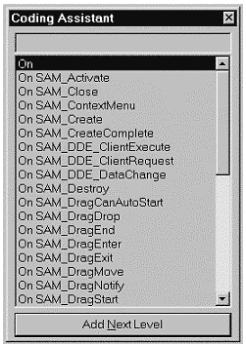
Control	**Comments**
Classic menu	For routine users, as it presumes knowledge of shortcuts, etc.; many actions or program

	modules can be converted to menu options.
Form window with pushbuttons	For inexperienced users, as available actions are always visible and don't need to be sought in menu structures.
Central MDI window	Fundamental modules (e.g., addresses, assignments) can be permanently visibly arranged; actions concerned with details (e.g., deleting data records) are only displayed as needed.

Help in Gupta

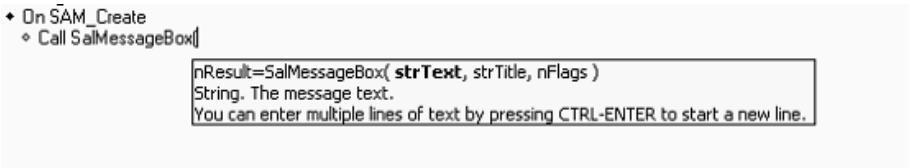
The menu option Help can be used to find help with functions, messages, etc.; furthermore, electronic manuals can be found in the Team Developer option *Books Online*.

The Coding Assistant can also be used when coding an application. The Coding Assistant always displays the functions or messages which can be inserted into the currently active location, and can be opened with the key combination Alt+2 (or from the Tools menu).



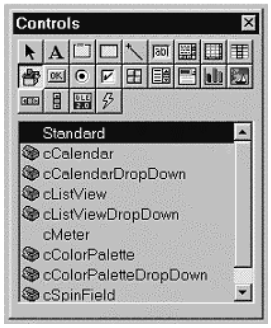
Illus.: Coding Assistant

Popup help is also integrated in Team Developer versions 3.0 and later. Context sensitive help is automatically displayed during coding showing the programmer the available options.



Illus.: Context sensitive help

Furthermore, an additional window, the so-called Controls window, can be displayed with the objects (opened with Alt+4 or from the Tools menu).



Illus.: Controls with available objects (top) and classes (below).

Chapter 1.2
An Introduction to Gupta Team Developer

In this chapter the programmer is brought into first contact with Gupta Team Developer. After general tips for using the software, examples are shown which illustrate various aspects of programming. Thereafter, the basics of SAL (Scalable Application Language) are described in detail. Lastly, tips about objects and classes as well as an introduction to the integration wizard are given.

Chapter 1.2.1
Standard Practice with Gupta Team Developer

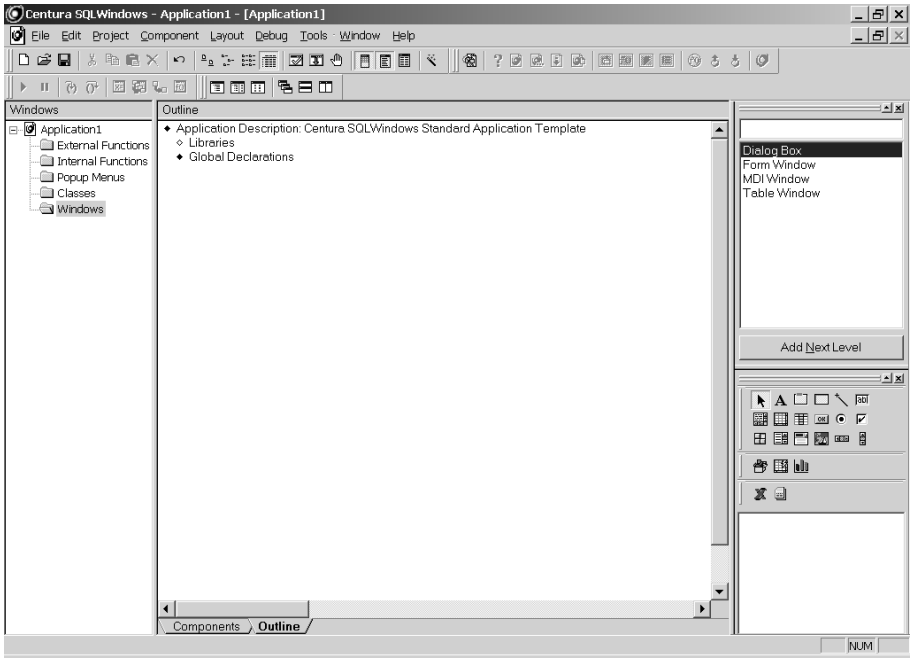
The programming process with Gupta usually takes the following steps:

- Creating a source code (manually, self-programmed, poss. using quick objects)
- Compiling, creating an EXE file
- Installing library files (content of DEPLOY directory) on target computer together with EXE file and any files for reports
- Installing the DB system
- Startup

Debugging and additional modifications are normally also necessary to fine tune the application to the customer's wishes. As mentioned above, brief, respective examples ensure that the material is easily understood. The examples have been specifically selected to be comprehendible with a minimum of effort (each example provides all necessary "grips" so that the beginner doesn't have to search for, say, variables which need to be defined somewhere). In addition – except in the chapter which is solely present for quick objects – as few prefab modules are used as possible because experience shows that the program code can thus be implemented in individual programs more quickly.

Chapter 1.2.1.1
The SQLWindows Startup Screen

The following illustration depicts the startup screen from SQLWindows as it appears after the program has been opened (a new Application has been started).



Illus.: SQLWindows startup screen (explanations below)

A few comments about the startup screen:

- The screen is divided into two main sections, called panes. The left pane runs objects which are included in the (current) application. The right pane offers details about the object selected (left pane).
- A new application consists of the sections external functions, internal functions, popup menus, classes, and windows.
- The right pane also shows the “assistants” which help during coding: controls (which includes the child objects which can be contained in a window), and the Coding Assistant, which displays messages or functions which could be used later.

The content of a new application depends on a file named Newapp.App, which is always read and copied when SQLWindows is started or a new application is created over File/New.

The File Newapp.App

The file Newapp.App is created during Gupta installation in the directory \Gupta\Templates\. It serves as a template (bare framework) for creating new applications. This means that every time the selection is made over File/New to create a new application, a copy of Newapp.App is created and added to main memory. This is then edited and saved under a new name.

Of course the template Newapp.App can be modified. This is a good idea in cases where recurring settings or objects (e.g., a MDI window with a certain menu control) are used. Then the elements don't need to be added each time when writing a new application. All that is necessary is to save a new application which has been written with the desired objects under the name Newapp.App.

Another variation is to create yet another file as template for new applications. Its settings can be made from the menu option Tools/Preferences.

Components

Components are already present in the template file Newapp.App. The extent and functionality of these components are almost unlimitedly variable. Not entirely, though, because a few components cannot be removed from an application.

The following overview lists the above mentioned components as they appear in Newapp.App after installation and offers brief descriptions of their contents.

Component	**Content**
External functions	DLL files containing C or assembler functions
Internal functions	Functions written by user in SAL
Popup menus	Prefab menu options for top level windows (e.g., typical Windows options like edit, copy)
Classes	Functions and quick object classes which are integrated over APL files or are defined in the source code itself
Windows	All top level or child windows (e.g., objects) that belong to the application

Every object (e.g., a new MDI window or a pushbutton) is added to the list of components. This list can be displayed by opening the individual sections.

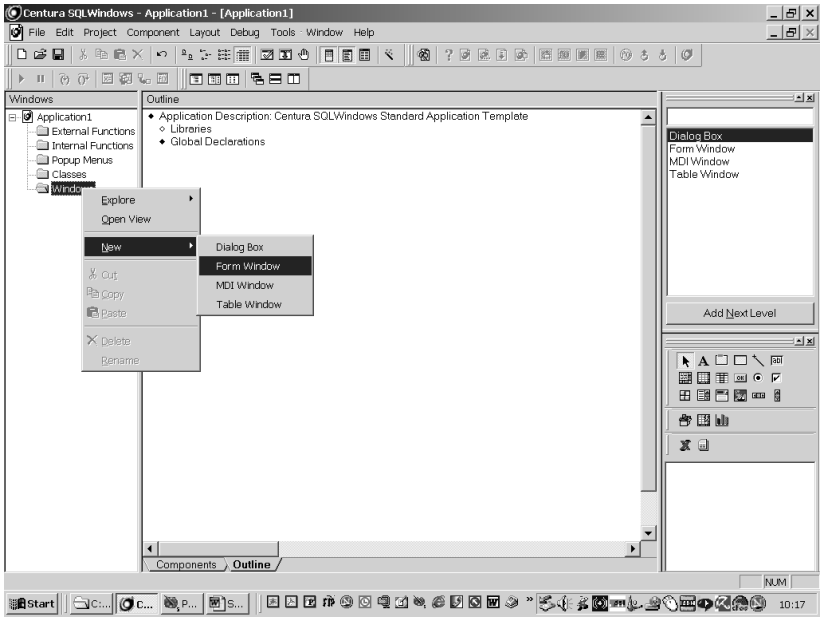
Using Controls and the Coding Assistant

Two examples explain the use of these tools.

Controls

Controls should be used here to add a pushbutton to a form window. Since the application does not yet feature a form window, however, the form window must first be added.

To add a form window to an application, click with the right mouse button on the entry “Windows”, select the option “New” and then “Form Window”.



Illus.: Adding a new form window to an application

Comments:

- The form window is initially unnamed. This is the first thing to enter; here, the name “frmCustomer”(names like “frm1” are unsuitable because complex applications often have10-20 form windows, making numbering relatively useless).
- A form window can be renamed later by right clicking it and selecting “Rename”.
- If the entry “(no name)” is not changed, an error message will appear.

- Top level windows (i.e., form windows table windows, dialog boxes, and to some extent also MDI windows) are always added to an application as described above.
- Which top level windows can be added depends on the associated libraries. Here, for instance, the quick tab window is not shown (and therefore cannot be added). To enable use of this special object (which will be described more completely later) the relevant library must first be associated [with the application. Note that libraries can only be added at the start of an application. First thing, then is to click the application name in the left pane, which displays, among other things, the option “Libraries”. Click this option to activate that section; right click “Libraries” and then “Add Next Level” to display a menu which includes the library named QCKTABS.APL. Clicking this name adds the library; thereafter, quick tab windows are available to the application.
- Beginners can also add top level windows easily with a wizard (menu “Component/Wizards” or the button with the magic wand). The programmer is then prompted through all absolutely necessary entries.

To add a pushbutton to the new form window:

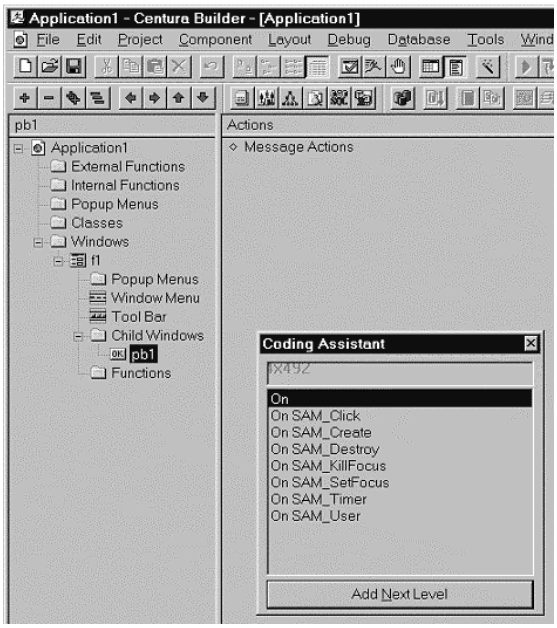
- First, the form window which will be furnished with a pushbutton must be selected; right click the form window’s name in the left pane and select “select in layout” from the menu. The newly added form window is now displayed in the right pane (shortcut: Ctrl+L).
- Next, the pushbutton is added to the form window (first check that the controls window is visible; if not, display it using Alt+4 or from the toolbar). Select the pushbutton object from the controls window by clicking on it once. Below, the word “standard” appears, meaning that currently only standard classes are available (this is sufficient at this time; later, other class names will be shown).
- The cursor now moves automatically to the form window, changes its appearance and indicates that clicking (on a particular spot in the form window) will add a pushbutton. The button’s size can also be set at this time by dragging the mouse (move the mouse to the right while holding down the mouse button).
- After the pushbutton has been added, the word “untitled” appears and can immediately be changed. Note that this word causes no error message because the name with which this pushbutton is targeted is actually “pb1” rather than “untitled”; this can be checked by double clicking the pushbutton and selecting “Object Name” from the menu. The button’s name can be changed at any time.

Now the pushbutton has been added. The Coding Assistant is used to assign an action to the pushbutton.

Coding Assistant

The Coding Assistant is used to associate an action with a pushbutton. First, the Coding Assistant must be started (Alt+2 or from the toolbar) and the following steps taken:

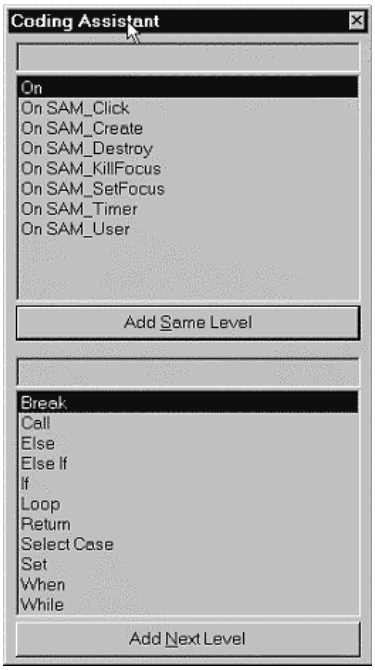
- First the pushbutton's so-called message actions must be defined. Open this by either right clicking the pushbutton and selecting the option "Explore Actions", or simply clicking the pushbutton and shifting over the "Actions" option to the message actions.



Illus*.: A pushbutton's message actions (with coding assistant)*

- The Coding Assistant now shows a list of entries which all begin with "On SAM_". These are the possible messages available for a pushbutton in this location (if Coding Assistant doesn't display the entries, it can be given a little help by hitting first the Insert key and then the Esc key on the keyboard). The entry On SAM_Click should now be chosen be double clicking it. This adds the line to the background screen. The Coding Assistant now changes its

appearance (displaying two windows) and content (the messages are now shown at the top and SAL statements below).



Illus.: Two-part Coding Assistant

- Now a function is invoked. This is done with the statement Call, by double clicking it. This word is then also added and the Coding Assistant now shows the functions of the SAL language (later, other, individual functions will also be displayed).
- The sought function is named SalMessageBeep(). It can be displayed by scrolling through the list or entering “Salme” in the input field. The function is then marked and can be added by double clicking it. Online help for this function can be used to confirm an entry’s function and is started by hitting F1.

Illus.: The function SalMessageBeep()

- As the word “Number” implies, starting the function requires a numerical value. Here, the value “-1” is entered, which creates the standard computer beep (details can be easily displayed by double clicking the function name and hitting F1; scrolling through help will reveal an example of this function).
- After the value has been entered, the program can be started with the F7 function key (a name must be entered at initial startup). If, after successful compilation, the pushbutton is pressed, an expressive “beep” will sound.

This small application can also, like every other windows program, be ended with the key combination Alt-F4 (or another pushbutton can be added that invokes the function SalQuit(), which ends the application. Coding this function will be described later).

A Few Important Menu Options in the Gupta Development Environment

A few menu options in the development environment are relatively important and are therefore briefly introduced here. During orientation with and later development of applications, one should occasionally check that, for instance, the