INM V12 Database

For Macromedia® Director®

Version 3.4

User Manual

Contents

License Agreement

PLEASE READ THIS LICENSE AGREEMENT CAREFULLY BEFORE USING INM V12 DATABASE. BY USING INM V12 DATABASE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS LICENSE AGREEMENT.

The enclosed computer program(s), license file and data (collectively, "Software") are licensed, not sold, to you by Integration New Media, Inc. ("Integration") for the purpose of using it for the development of your own products ("Products") only under the terms of this Agreement, and Integration reserves any rights not expressly granted to you. Integration grants you no right, title or interest in or to the Software. The Software is owned by Integration and is protected by International copyright laws and international treaties.

1. License.

(a) You may install one copy of the Software on a single computer. To "install" the Software means that the Software is either loaded or installed on the permanent memory of a computer (i.e., hard disk). This installed copy of the Software may be accessible by multiple computers; however, the Software cannot be installed on more than one computer at any time. You may only install the Software on another computer if you first remove the Software from the computer on which it was previously installed. You may not sublease, rent, loan or lease the Software.

(b) You may make one copy of the Software in Machine-readable form solely for backup purposes. As an express condition of this Agreement, you must reproduce on each copy any copyright notice or other proprietary notice that is on the original copy supplied by Integration.

(c) You may permanently transfer all your rights under this Agreement to another party by providing to such party all copies of the Software licensed under this Agreement together with a copy of this Agreement and the accompanying written materials, provided that the other party reads and agrees to accept the terms and conditions of this Agreement and that you keep no copy of the Software. If the Software is an update, any transfer must include the update and all prior versions.

(d) Your license is limited to the particular version (collectively "Version") of the Software you have purchased. Therefore, use of a Version other than the one encompassed by this License Agreement requires a separate license.

(e) The Software contains a license file (.LIC) which is subject to the restrictions set forth above and may not be distributed by you in any way. However, Integration grants you a royalty-free right to reproduce and distribute the files named "V12-Database.XTR" and "V12-Database.X32" (collectively, "Runtime Kit") provided that (i) you distribute the Runtime Kit only in conjunction with and as part of your own Products; (ii) own a license for the specific Version of the Software that contains the Runtime Kit; (iii) agree to indemnify, hold harmless and defend Integration from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your Products with the Runtime Kit.

(f) Any third party who may distribute or otherwise make available a product containing the INM V12 Database Runtime Kit must purchase its own INM V12 Database license.

(g) Any third party who will use the INM V12 Database Runtime Kit in an authoring environment must purchase his own INM V12 Database license.

2. Restrictions. The Software contains trade secrets in its human perceivable form and, to protect them, you may not MODIFY, TRANSLATE, REVERSE ENGINEER, REVERSE ASSEMBLE, DECOMPILE, DISASSEMBLE OR OTHERWISE REDUCE THE SOFTWARE TO ANY HUMAN PERCEIVABLE FORM. YOU MAY NOT MODIFY, ADAPT, TRANSLATE, RENT, LEASE, LOAN OR CREATE DERIVATIVE WORKS BASED UPON THE SOFTWARE OR ANY PART THEREOF.

3. Copyright notices. You may not alter or change Integration's copyright notices as contained in INM V12 Database. You must include (a) a copyright notice, in direct proximity to your own copyright notice, in substantially the following form "Portions of code are Copyright (c)1995-2005 Integration New Media, Inc."; and (b) place the "Powered by V12" logo on the packaging of your Products; or (c) place the "Powered by V12" logo within your Products in the credits section.

4. Acceptance. INM V12 Database shall be deemed accepted by you upon delivery unless you provide Integration, within two (2) weeks therein, with a written description of any bona fide defects in material or workmanship.

5. Termination. This Agreement is effective until terminated. This Agreement will terminate immediately without notice from Integration or judicial resolution if you fail to comply with any provision of this

Agreement. Upon such termination you must destroy the Software, all accompanying written materials and all copies thereof, and Sections 7 and 8 will survive any termination.

6. Limited Warranty. Integration warrants for a period of ninety (90) days from your date of purchase (as evidenced by a copy of your receipt) that the media on which the Software is recorded will be free from defects in materials and workmanship under normal use and the Software will perform substantially in accordance with the manual. Integration's entire liability and your sole and exclusive remedy for any breach of the foregoing limited warranty will be, at Integration's option, replacement of the disk, refund of the purchase price or repair or replacement of the Software.

7. Limitation of Remedies and Damages. In no event will Integration, its parent or subsidiaries or any of the licensers, directors, officers, employees or affiliates of any of the foregoing be liable to you for any consequential, incidental, indirect or special damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of business information and the like), whether foreseeable or not, arising out of the use of or inability to use the Software or accompanying written materials, regardless of the basis of the claim and even if Integration or an Integration representative has been advised of the possibility of such damage. Integration's liability to you for direct damages for any cause whatsoever, and regardless of the form of the action, will be limited to the greater of US $350.00 or the money paid for the Software that caused the damages.

THIS LIMITATION WILL NOT APPLY IN CASE OF PERSONAL INJURY ONLY WHERE AND TO THE EXTENT THAT APPLICABLE LAW REQUIRES SUCH LIABILITY. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

8. General. This Agreement will be construed under the laws of the Province of Quebec, except for that body of law dealing with conflicts of law. If any provision of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this Agreement will remain in full force and effect.

9. The parties acknowledge having requested and being satisfied that this Agreement and its accessories be drawn in English. Les parties reconnaissent avoir demandé que cette entente et ses documents connexes soient rédigés en anglais et s'en déclarent satisfaits.

Introduction

Welcome to INM V12 Database (INM V12 Database), the most versatile and user-friendly cross-platform database management Xtra for Macromedia Director®, Authorware® and now available for Flash®.

INM V12 Database for Director

INM V12 Database was originally designed in 1996 to be used specifically with Director. It extends Director's features and helps you speed up the development of your multimedia titles. You will discover many benefits in using INM V12 Database to create interactive applications such as electronic catalogs, storybooks, kiosks, training material, sales material, games, and more. Used as a back-end to your multimedia projects, INM V12 Database lets you efficiently manage text, numeric data, dates, images, sound clips as well as any type of media that Director can store in its members.

INM V12 Database enables you to provide advanced functionality to your end-users while bringing down your development and maintenance costs.

INM V12 Database is very flexible and scalable. It can be used in a wide range of applications, from simple projects where Lingo Lists and FileIO have become difficult to manage, to true database-driven applications.

INM V12 Database for Director is available in *Light* and *Regular* editions. Both *Regular* and *Light Editions* are fully compatible with each other, so you can start using INM V12 Database *Light Edition*, and later easily upgrade to INM V12 Database *Regular Edition*. For limitations on the Light Version, see Capacities and Limits. A comparison chart for INM V12 Database Regular vs. Light is available on Integration New Media's web site at: http://www.INM.com/products/v12director/info/.

INM V12 Database Online Companion

INM V12 Database Online Companion (available for Director only) is a powerful add-on that extends INM V12 Database's capabilities, allowing your applications to dynamically connect to V12 databases via any TCP/IP network, be it intranet, extranet or the Internet. Applications that use INM V12 Database Online can be packaged as either Director Projectors or Shockwave movies.

Lingo programmers will find the INM V12 Database Online Companion easy to learn and use, especially if they have previously experimented with alternatives such as interfacing custom CGIs to an HTTP server and/or dealing with databases through third party middleware. Also, since the Online Companion is a natural extension to INM V12 Database, projects that already use INM V12 Database can be very easily converted to take advantage of the Online Companion.

To learn more about the INM V12 Database Online Companion or get a free evaluation version, visit the product website at: http://www.INM.com/products/v12online/

INM V12 Database for Flash

Thanks to the capabilities of Macromedia Flash and Director to communicate and exchange data, INM V12 Database can now be used by Flash developers via a projector. This projector wraps a Flash .SWF file so that Flash developers do not need Director, and they can access a V12 Database via a very similar API as Director users. For more information and to download a trial version visit:

http://www.INM.com/products/v12Flash/

About This Manual

This manual is organized to help you get the information you need efficiently. The *Using Xtras* section deals with basics concerning Xtras and the *Database Basics* gives an overview of databases. *Using INM V12 Database* outlines the process of creating a V12 database-driven project. In subsequent sections you will learn how to prepare data, create a database and import data. The section entitled “Step 5: Implementing the User Interface” shows you how to use the methods available in INM V12 Database.

The next sections cover the integration of INM V12 Database with Macromedia Director—here you can get a sense of how INM V12 Database can be helpful to your projects. The Appendices deal with very specific issues such as capacities and limitations, end-user delivery, portability, and special features.

For a complete description of all the INM V12 Database methods consult the sections named **V12 Database Methods** and **V12 Database Error Codes** in this manual.

Where to start

Before browsing through this User Manual we recommend that you look at the **First Steps** tutorial. This introductory tutorial contains example scripts, and sample projects to help you get started using INM V12 Database both using Lingo and Behaviors. After completing the tutorial you will have a better basis for using this manual, while planning your own V12 project. Download the First Steps for INM V12 Database from our website at:
http://www.INM.com/products/v12director/first-steps/.

You may also benefit from browsing through the V12 Sample movies, available for FREE on INM’s website at: http://www.INM.com/products/v12director/demos/.

Please make sure you understand INM V12 Database's license agreement before proceeding. The full license agreement is at the beginning of this user manual (see License Agreement).

Do I really need to master Lingo to use INM V12 Database?

How comfortable do you need to be with Lingo to use INM V12 Database efficiently? The answer varies according to the complexity of your projects.

Simple projects require **no knowledge of Lingo at all**. If your project uses a single database and shows one record at a time on Director's stage, chances are that you can implement it using the freely available *INM V12 Database Behaviors Library* only. If you don't need any more functionality than the Behavior Library provides, then no Lingo programming will be required.

For more advanced projects, INM V12 Database's comprehensive Lingo interface requires knowledge of Lingo. However, programming guidance, such as checking the number and type of parameters, is provided wherever possible.

In a nutshell, before coding your project using INM V12 Database, you need to know the following basic Lingo concepts:

- Local and global variables,
- Control structures (`if` statements, `repeat` loops, etc.),
- Handlers
- Object instances (this is covered in detail later in the Using Xtras section of this manual)

The INM V12 Database Behaviors library features only the basic essential features of INM V12 Database's functionality. Before you commit yourself to using the INM V12 Database Behaviors library in your project, you may first want to ask Support@INM.com or other INM V12 Database users on V12-L (http://www.INM.com/support/list) for advice.

You're not alone!

Whether you are looking for a quick answer or in-depth information, there are many resources available online and offline to help you. There are many, many registered owners of INM V12 Database that are sharing concepts and strategies online. And, you can also call INM's customer support, developer assistance or advanced services team to help you with any specific problems you may have. The following resources may be very helpful:

V12-L discussion list

On the V12-L Discussion List you will find developers at every level of expertise, and in a wide variety of specialties in the multimedia arena. This friendly group is the perfect place to bounce ideas around with other INM V12 Database developers. Sign up at http://www.INM.com/support/list/

Tech notes

Tech Notes discuss issues asked by INM V12 Database users that are too specialized or specific to be detailed in this user manual. Many of the Tech Notes are accompanied by open-source Director movies that you can download and experiment with. Check them out at: http://www.INM.com/support/v12director/technotes/

Other online resources

Macromedia's web site, at http://www.macromedia.com/support, is also a possible source of information. It contains, among other things, directions on how to subscribe to Macromedia's support Newsgroups (the NNTP, or “news” server is "forums.macromedia.com").

You may want to check alternate Internet resources such as

Director-online-User Group (http://www.director-online.com/),

UpdateStage (http://www.updatestage.com/),

Maricopa (http://www.mcli.dist.maricopa.edu/director),

and the Lingo User Journal (http://www.penworks.com/).

Customer support and developer assistance

If you need additional assistance, INM’s experienced team will be happy to help.

Customer support is available from 9:00 am to 5:00 pm EST, Monday through Friday by email to support@INM.com or by phone at:

+1 514 871 1333, Option 6.

Priority will be given to registered INM V12 Database users. Customer support covers:

- Helping you to understand INM V12 Database, clarifying specifications.
- Supplying you with sample scripts for generic concepts.
- Providing useful tips.

Where Customer Support stops, Developer Assistance begins. If you are familiar enough with INM V12 Database, but want to take your project to a more sophisticated level, Developer Assistance is for you. Our team of programmers can help you discover easier ways to take advantage of databases in your multimedia projects. Here are just some of the services we offer:

- Project design, data structure analysis, planning
- Technical guidance throughout the various steps of your project
- Troubleshooting and debugging your scripts
- Optimization (how to obtain superior performance)
- Assistance with other Xtras, custom development of Xtras

You can think of Developer Assistance as an additional team member, or members that you can add on to your project team. They can fulfill a small or large role on your team, depending on your needs.

Typographic conventions in this manual

Important terms, such as the names of methods, are in **bold**.

```
Sample code is indented and printed in this font.
```

Although the sample scripts throughout this manual contain both upper and lower case characters, INM V12 Database is *not* case sensitive. This applies to the methods names, the parameters as well as to the actual data. They are described using mixed case in order to improve readability and facilitate debugging, and we recommend you adopt a similar strategy in your Lingo programming. It really does help to improve the readability of your Lingo scripts!

Note: Special annotations and tips are enclosed in the sidebar, like this one.

Welcome to INM V12 Database

Welcome to INM V12 Database, the most powerful and user-friendly cross-platform database management Xtra for Macromedia Director on Mac OS 9, Mac OS X and Windows.

If you are familiar with other database management systems, you will find INM V12 Database very easy to use. If you are only vaguely familiar with database management systems, the next few sections will give you an overview of what you need to know to help you get started with INM V12 Database.

System requirements for running INM V12 Database

Macintosh version

- Mac Classic version 8.x+
- Mac OS X version 10.2 +

On the Macintosh, INM V12 Database (and any other Xtra) will share the same memory partition as Macromedia Director.

For simple database applications, you probably do not need to change the memory partition allocated to Director or to projectors generated by Director. For more advanced development, you may need to increase the memory partition. See your operating system manual for details.

In general, we recommend that your Director application's memory partition be set at the maximum you can afford to give to it, with enough RAM memory reserved for any other applications you may need to run at the same time as Director.

We also recommend that you allocate an absolute minimum of 8 megabytes of RAM to the projector's minimum RAM requirements. You may also want to ensure that Director's preference for "Use Temporary Memory" setting is set before creating your projector (this will ensure that if the projector does run out of memory, it will start using free System memory and dynamically increase the memory allocated to the application.

Tip: INM recommends that you always thoroughly test your final project using the minimum standard equipment you have determined for your application, on all supported operating system platforms.

This process will also help you confirm that your product functions correctly within the minimum memory requirement you have recommended to your users.

Windows version

- Windows 95, 98, Me, NT4, 2000 and XP

Windows uses a Virtual Memory scheme, which dynamically allocates memory to applications. This means that an application can "borrow" as much memory as needed from the Operating System. It also means that Windows shows unpredictable behaviors when it is short of memory. Try to establish the minimum equipment requirements of your project as conservatively as possible.

Always thoroughly test your project on the minimum required equipment you have determined for your application.

Macromedia Director

Macromedia Director version 8, 8.5, MX or MX 2004 is required.

Installing INM V12 Database

The name of this Xtra is **V12-Database.X32** on Windows, and **V12-Database.XTR** on the Mac (the files are named the same for Mac OS 9 and Mac OS X, but they are different Xtras and are located in separate folders within the Mac download package).

To install the INM V12 Database Xtra in your authoring environment:

- Make sure that Director is closed.
- Move the INM V12 Database Xtra to the Xtras folder located in the same folder as Director (for Director MX 2004 the Xtras folder is in the folder named Configuration).
- Start Director.

To confirm that INM V12 Database is properly installed, check the Xtras menu in Director. You should see "INM V12 Database" in the Xtras menu.

Version history

INM V12 Database version 1.0 was released in 1996 as both a XObject and Xtra for Macromedia Director 4 and 5. It was essentially meant to serve as an advanced data management system for Director titles with elaborate user interfaces delivered on CD-ROM, such as games and virtual workshops.

INM V12 Database Xtra version 2.0 was released in early 1998. It focused on making database technology easier to learn and use by Director users. It added features that better suit projects such as electronic catalogs, electronic books, template-based movies, etc. Some of these features are: full-text indexing, simplified database creation, data binding, styled text management, a Behaviors library, etc.

INM V12 Database Xtra version 3.0 was released in July 1999. It focused on extending INM V12 Database to multi-user environments and to better serve Internet applications. Minor releases since version 3.0 have, for the most part, addressed bugs and compatibility issues between different versions of Director.

Release notes from previous versions

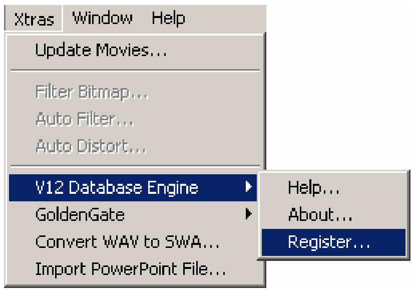
For details on what changed in previous releases of INM V12 Database, please visit the Release-History pages in the support section of Integration New Media's web site: http://www.INM.com/support/v12director/release-history/

How to register your INM V12 Database license

Free evaluation copies of INM V12 Database are available on Integration New Media's web site (http://www.INM.com/V12director/) along with full documentation and sample movies. You can download those files today and evaluate them in your project without purchasing an INM V12 Database license.

The evaluation copy of INM V12 Database is not limited in any way; it only displays a splash screen upon startup. To avoid seeing the splash screen, you must purchase an INM V12 Database license (or as many as required by the INM V12 Database license agreement). As a licensee, you are given a license key that you must enter into the **Enter Key** dialog box, from within Director.

Click **Xtras > INM V12 Database > Enter Key…** from Director's menu.



Enter your name, company name and the license key you received by e-mail upon purchasing your license. Once your license key is entered, all new databases you create are automatically licensed and do not show a splash screen. Existing databases created in trial mode are licensed as soon as they are opened in **ReadWrite** mode on that machine.

Note: Existing V12 databases must be opened once in ReadWrite mode to be licensed. If you open them in ReadOnly mode only or from a CD-ROM, they **cannot** be licensed and the splash screen will continue to appear on computers that do not have the license file, even if you purchased a license. INM V12 Database returns a warning when opening unlicensed databases in such circumstances.

The only way to test if your V12 Database file is licensed for playback and distribution is to **test it on a machine that does not contain the license key**.

Additional Tools for Working with INM V12 Database

To assist you in developing your V12 Database projects, Integration New Media has developed some productivity tools that are available Free from the *Free Tools* section of our website:
http://www.INM.com/products/v12director/tools/.

INM V12 Database Editor

The INM V12 Database Editor is an application that helps you create, edit and browse the contents of V12 databases. It also comes with a variety of predefined V12 database templates to help you get started quickly. The INM V12 Database Editor is discussed in this manual in the section entitled, Using the INM V12 Database Editor, a sub-heading of Step 3: Create a Database.

INM V12 Database Behaviors Library

The INM V12 Database Behaviors Library is set of Behaviors that allow you to quickly implement a simple application interface using a V12 database. If your project uses a single database and shows one record at a time on Director's stage, chances are that you can implement it using the INM V12 Database Behaviors Library only. You can read more on the *INM V12 Database Behaviors Library* in it's accompanying manual.

INM V12 Database Help Files

Integrated help files are available to assist you while working in Director. These files are accessed from Director's Xtras menu to provide quick reference for the INM V12 Database methods, error codes and Behaviors. Download the Help MIAW (movie in a window) from: http://www.INM.com/support/v12director/manuals/.

Using Xtras

This section deals with Xtras and how they are used in Macromedia Director. The INM V12 Database Xtra is used as an example throughout the manual. You will be introduced to the basic steps involved in using INM V12 Database successfully before you actually begin to work with INM V12 Database.

This section covers:

- Making an Xtra available to Director
- Creating an Xtra instance
- Verifying whether the instance was successfully created
- Using the Xtra instance
- Closing the Xtra instance
- Where to get additional Documentation
- Using Xtras with Shockwave

What is an Xtra?

Xtras are components (alternatively know as extensions, add-ons, or plug-ins) that add new features to Macromedia Director. Many of Director's own functions are implemented as Xtras. Xtras for Windows must have an .X32 file extension, as in "V12-Database.X32".

Making the INM V12 Database Xtra available in authoring and runtime modes

Xtras are designed to be opened automatically by Director (in authoring mode) or by your Projector (in runtime mode, also called *playback* mode). The INM V12 Database Xtra must be placed in the **Xtras** folder, located either in Director's folder or the same folder as your Projector.

If you are delivering an end-product using Shockwave, then the way Xtras are installed and located is a bit different than with Projectors and in the authoring environment. See Using INM V12 Database with Shockwave for more information.

Checking for available Xtras

You can learn which Xtras are available in your copy of Director by typing the following command in the Message Window:

```
ShowXlib
```

If INM V12 Database is installed, you should see V12dbe and V12table listed in `ShowXlib`'s output, as well as all other available Lingo Xtras.

Creating an instance of the INM V12 Database Xtra

A mandatory first step before using a Lingo Xtra is to create an instance of it. This is how a Lingo Xtra comes to life and, from then on, you can use its methods.

Call `New` to create an instance of the INM V12 Database Xtra. Generally, you store a newly created Xtra instance in a global variable for future use. The following example uses the `New` method of the database Xtra.

Example:

```
gDB = New(Xtra"V12dbe", the MoviePath&"myBase.V12", "Create",
   "myPassword")
```

Checking if *New* was successful

You should always ensure that the Xtra was created successfully immediately after calling `New`. `New` can fail for many reasons, such as a lack of free memory or as a result of misplaced files.

Example:

```
if NOT ObjectP(gDB) then alert "Could not create Xtra instance"
```

> **Note:** This is a generic approach and works with all Xtras. In INM V12 Database, the preferred way to check for errors is the V12Status() method. See Checking the Status of the Last Method Called in this manual.

Using the Xtra instance

Once the preliminary steps have been executed, you can start using the Xtra instance of your database for creating tables, fields and indexes, or for using an existing database. **Methods** of the Xtra need to be called to perform these operations. By convention, INM V12 Database method names begin with the letter `m` such as `mGetfield` and `mSelect` (with a few exceptions such as `New`, `V12Error` and `V12status`).

This example shows the structure of the database referred to by `gDB` in the message window:

```
put mDumpStructure(gDB)
```

> **Note:** In order to learn which methods are supported by an Xtra, use the Xtra's built-in documentation. See Basic documentation below.

Closing the Xtra Instance

When the Xtra instance has completed its function and is no longer required, close it by calling the mClose() method and then setting the variable that refers to it to *0*. Closing an Xtra performs mandatory housekeeping tasks and closes unneeded files. It also frees the memory occupied by the Xtra. All Xtra instances created with `New` must be ultimately destroyed using mClose() and set to *0* once they are no longer needed.

Example:

```
gDB.mClose()
gDB = 0
```

Note: If a INM V12 Database Xtra instance is not properly closed, the file it refers to remains open and cannot be re-opened unless the computer is restarted. In some cases, the database could become corrupted.

Note: Up until Director MX 2004, simply setting the database variable to zero would close the database instance. However, with Director MX 2004, you must explicitly close the table and database instances using the mClose() method.

Dealing with pathnames

The `New` method in V12dbe requires that you specify the name of the INM V12 Database file you want to create or open. If only a file name is specified, the file is assumed to be located in the same folder as Director or the Projector.

If you are using Shockwave, INM V12 Database is limited to looking for the database in the standard application cache folder. Example:

```
gDB = New(Xtra"V12dbe", "myBase.V12", "Create", "myPassword")
```

assumes that "myBase.V12" is in the same folder as Director or the Projector. This is equivalent to:

```
gDB = New(Xtra"V12dbe", the applicationPath & "myBase.V12", "Create",
   "myPassword")
```

Most of the time, however, placing the database file in the same folder as the *movie* that uses it is more convenient. Use `the MoviePath` Lingo function to get the current movie's folder.

Example:

```
gDB = New(Xtra"V12dbe", the MoviePath & "myBase.V12", "Create",
   "myPassword")
```

Passing parameters to the INM V12 Database Xtra

As in any programming language (including Lingo), functions, commands and methods require a certain number of parameters. For example, in Lingo, the `Go to frame` command expects one parameter: the destination frame identifier. Likewise, the `getAt(theList, pos)` function expects two parameters: *list* and *position*.

While the two aforementioned examples require exactly one and two parameters respectively, some commands and functions offer more flexibility by accepting optional parameters. For example, in Lingo, the `Beep` command requires one parameter: the number of beeps. However, if that parameter is omitted, Lingo assumes that one beep is required.

Xtras offer the same mechanism: some methods require an exact number of parameters (*fixed number of parameters*), others assume default values if parameters are omitted (*variable number of parameters*). Each of these methods can be easily identified in the Xtras built-in documentation explained below (see Basic documentation).

Basic documentation

In Director, Xtras contain a built-in mechanism that provides documentation for Lingo developers. In the Message Window, type:

```
put interface(Xtra "V12dbe")
```

where Xtra "V12dbe" is the name of the Xtra library, not of an instance of the Xtra.

The above command returns the following Xtra description:

```
-- part of INM V12 Database
-- ©Integration New Media, Inc. 1995-2005
-- Please check the on-line help in the Xtras/INM V12 Database menu
new object me, string databasename, string openmode, *
 mBuild object me, *
mCloneDatabase object me, string databasename
mClose object me
mCreateField object me, string tablename, string fieldname, *
mCreateFullIndex object me, string tablename, string fieldname, *
mCreateIndex object me, string tablename, string indexname, string isunique,
string fieldname, string order, *
mCreateTable object me, string tablename
mCustom object me, *
mDeleteTable object me, string tablename
mDumpStructure object me, *
mEditDBStructure object me
+ mError object xtraRef, *
+ mFixDatabase object xtraRef, string databasename, string newdatabasename
mFlushToDisk object me
mGetPropertyNames object me, *
mGetProperty object me, string property
mGetRef object me
mPackDatabase object me, string newdatabasename
mReadDBStructure object me, string inputtype, string source, *
mRenameField object me, string tablename, string oldfieldname, string newfieldname
mSetPassword object me, string oldpassword, string newpassword
mSetProperty object me, string property, string value
+ mStatus object xtraRef
mUpdateDBStructure object me
* V12Download string url, string dest, string password, string
completion_callback_handler, string status_callback_handler, any ref
* V12DownloadInfo string url, any info, string callback_handler, any ref
* V12Error *
* V12ErrorReset
* V12Status

+ mXtraVersion object xtraRef
```

Methods that expect a fixed number of parameters are those for which each parameter is listed. Methods that accept a variable number of parameters are those followed by a *****.

Following are a few examples:

```
mFlushToDisk object me
```

means that the mFlushToDisk method requires exactly one parameter: the database instance.

```
mSetProperty object me, string property, string value
```

means that mSetProperty requires three parameters: the database instance, the property (a string) and the value of the property (a string).

```
mDumpStructure object me, *
```

means that mDumpStructure requires at least one parameter, and possibly more (indicated by the asterisk). You must refer to the documentation of this method to know what additional parameters are accepted, and when.

```
+ mXtraVersion object xtraRef
```

the leading "+" sign means that mXtraVersion is a static method - a method that can be used with a database instance (i.e. mXtraVersion(gDB)) and a database library instance (i.e. mXtraVersion(Xtra "V12dbe")). Static methods are seldom used in INM V12 Database.

```
* V12Status
```

the leading "*" means that V12status is a global method - a method that can be used at any time, regardless of Xtra instances. It only requires that the Xtra be present when that function is called.

For a complete listing of the INM V12 Database methods, see V12 Database Methods section of this manual.

Using INM V12 Database with Shockwave

Director 7 and greater have the ability to automatically download Xtras via the Internet, if they are not present in the user's computer. This functionality is open to use in Shockwave Projectors and Shockwave movies.

Due to the fact that you are using INM V12 Database in your project, you need to ensure that all users of your project have the Xtras properly installed and running before executing any of the INM V12 Database functions.

Macromedia has implemented an auto-downloading functionality into Shockwave, which utilizes the Verisign secure download process.

If you want to use Shockwave as your delivery platform, you should plan your project's implementation with this in mind. Be prepared to address the auto-installation issue for the V12 Database Xtra and any other third party or custom Xtras you may be using in your project.

More detailed information on using Xtras with Shockwave is described in this manual in the section entitled Shockwave Projectors and Movies.

Verisign

Auto-downloading Xtras use the Verisign secure download certificate to ensure the least amount of risk to the end user's computer. INM V12 Database is digitally "shrink-wrapped" to let your customers know that it is safe to install it on their computers. For mote details, see http://www.verisign.com.

Database Basics

Overview

If you want a clearer understanding of what a database is and does, we recommend that you read this section. The following sections deal with database basics:

- what is a database,
- records, fields and tables,
- indexes and full-text indexes, keen
- flat and relational databases,
- field types,
- selection, current record, and search criteria.

What is a database?

A database is a collection of information that can be structured and sorted. A telephone book is an example of a hardcopy database, and government statistical records are examples of electronic databases. Database management programs such as INM V12 Database provide many advantages over hardcopy databases. Unlike a telephone directory, where you can look up data that is sorted in alphabetical order only, database management programs allow you to change the way you sort and view information. Moreover, you can find, modify and update information quickly and easily.

Records, Fields and Tables

An entry in a database is called a **record**.

Each record consists of pieces of information called **fields**.

All records are stored in a **table**.

Note: Some database management systems refer to fields as columns and to records as lines or rows. In INM V12 Database, the terms remain `fields` and `records`.

For example, data entry in an address book typically consists of seven pieces of information called `fields`: last name, first name, street address, city, state, zip code and phone number. All the information relevant to one person makes up one `record`. The collected records make up the `table` and are contained in a `database` file. Entries below are typical of those found in an address book:

This is an example of a **table**:

Last Name	First Name	Address	City	State	Zip	Phone	<-- These are **fields**
Jordan	Ann	6772 Toyon Court	San Mateo	CA	94403	349-5353	<-- 1st **record**
Brown	Charles	30 Saxony Ave.	San Francisco	CA	94115	421-9963	<-- 2nd **record**
Pintado	Jack	22 Hoover Ave.	Bowie	MD	20712	731-5134	<-- 3rd **record**
Van Damme	Lucie	87 Main St.	Richmond	VA	23233	315-3545	<-- 4th **record**
Peppermint	Patty	127 Big St.	Lebanon	MO	92023	462-6267	<-- etc...

Indexes

In a telephone directory, information is indexed by last name - a typical way to search for a telephone number. There are directories that index information by order of phone number or address, but such static directories sort information in only one specific predetermined order.

INM V12 Database, by contrast, allows you to determine how you want to sort information by defining one or more **indexes** in a table. When a field is indexed, INM V12 Database creates an internal list that can be used to sort and search quickly the data it contains. Non-indexed fields can also be searched and sorted, but at a slower speed.

In this example, the address book entries are listed according to an index of the first name field and sorted in ascending order (A to Z), thus appearing in alphabetical order by first name.

Last Name	First Name	Address	City	State	Zip	Phone
Jordan	Ann	6772 Toyon Court	San Mateo	CA	94403	349-5353
Brown	Charles	30 Saxony Ave.	San Francisco	CA	94115	421-9963
Pintado	Jack	22 Hoover Ave.	Bowie	MD	20712	731-5134
Van Damme	Lucie	87 Main St.	Richmond	VA	23233	315-3545
Peppermint	Patty	127 Big St.	Lebanon	MO	92023	462-6267

Compound indexes

A *compound index* — or *complex index* —organizes entries composed of two or more fields, as opposed to simple indexes — or indexes, for short — which organize single-field entries. Compound indexes are useful to determine the sorting order of records when some fields contain identical values.

In this example, three records share the same last names (Cartman). Indexing the field `LastName` alone would certainly force Last Names to be properly ordered. But this would not determine the order in which the Cartman's are sorted.

Last Name	First Name	City	State	Zip
Brown	Charles	San Francisco	CA	94115
Cartman	Wendy	San Mateo	CA	94403
Cartman	Lucy	Richmond	VA	23233
Cartman	Eric	Lebanon	MO	92023
Pintado	Jack	Bowie	MD	20712

If you want your records sorted by Last Name, and by First Name in case of identical Last Names, you define a compound index on the fields `LastName` and `FirstName`. The sorted result would then be:

Last Name	First Name	City	State	Zip
Brown	Charles	San Francisco	CA	94115
Cartman	Eric	Lebanon	MO	92023
Cartman	Lucy	Richmond	VA	23233
Cartman	Wendy	San Mateo	CA	94403
Pintado	Jack	Bowie	MD	20712

If you want them sorted by Last Name, and then by State in case of identical Last Names you define a compound index on the fields `LastName` and `State`. The sorted result would then be:

Last Name	First Name	City	State	Zip
Brown	Charles	San Francisco	CA	94115
Cartman	Wendy	San Mateo	CA	94403
Cartman	Eric	Lebanon	MO	92023
Cartman	Lucy	Richmond	VA	23233
Pintado	Jack	Bowie	MD	20712

Up to twelve fields can be declared in a single compound index in INM V12 Database.

Full-text indexing

Defining an index on a field allows for quick sorting and searching *of the first few characters* of a field. In some applications – typically when fields contain extensive information – you need to search for words that appear *anywhere* in a field efficiently. This is where you need to define a full-text index, or **full-index** for short, on that field. A full-index is an index defined on every single word of a field.

Last Name	First Name	Publication Title
Jordan	Ann	Soups and Salads for Dummies
Brown	Charles	The Hunchback of the Empire State Building
Pintado	Jack	Bounds on Branching Programs
Van Damme	Lucie	Natural and Artificial Intelligence
Peppermint	Patty	Mastering Soups in 32,767 Easy Lessons

In this example, looking for the word "Soup" in the Publication Title field requires a full-index for optimal search performance. If no index is defined on the Publication Title field, the same result can be achieved, but with a slower performance. If a regular index is defined on the Publication Title field, publications that start with the word "Soup" can be quickly located, but publications that contain that word somewhere in the middle of the field require more time. Full-indexes apply only to fields of type `string`, including those that contain styled text (see Field types, International support and Styled text).

Note: Each index takes up disk space, so it is not recommended that all fields be indexed. Full-indexes require much more space than regular indexes. Indexed fields should be limited to those most likely to be searched and/or sorted.

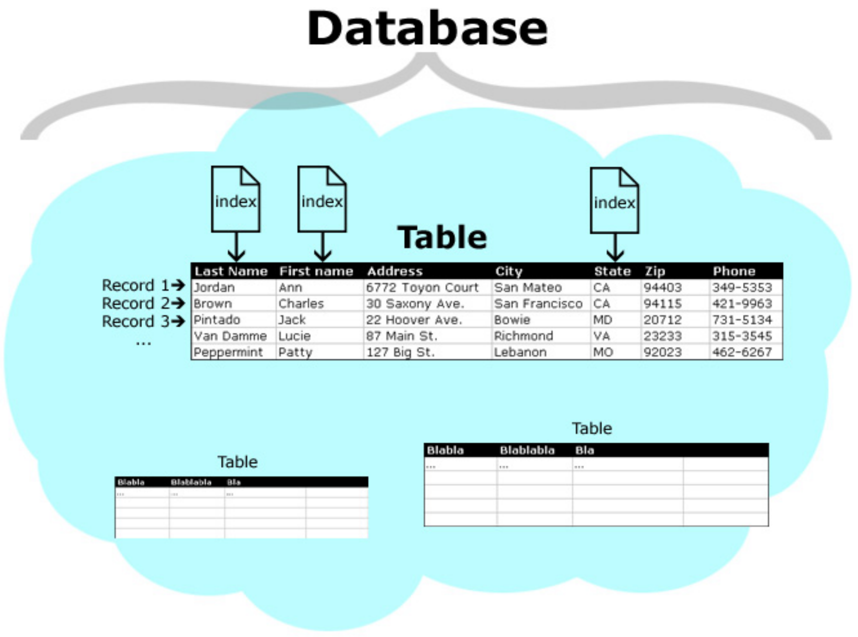
Defining full-text indexing options

For optimal full-text search efficiency, some level of control is required on the way it is performed. For example, indexing trivial words such as "and", "or", "the", etc. (or equivalent words that appear frequently in your application's language) is useless as most records would contain one or more occurrences of those words.

Likewise, some applications or languages require that digits be full-indexed whereas others would prefer to ignore them. INM V12 Database enables you to fine-tune the behavior of the full-indexes by allowing for the definition of **Stop Words** (words that must be ignored), **Delimiters** (characters that delimit word boundaries) and **MinWordLength** (the size of the shortest word that must be considered for full-indexing). These values must be defined before the structure of the database is created.

Database

A table, its fields and the indexes defined are stored in a **database**. A database can contain one or more such tables.



Last Name	First name	Address	City	State	Zip	Phone
Jordan	Ann	6772 Toyon Court	San Mateo	CA	94403	349-5353
Brown	Charles	30 Saxony Ave.	San Francisco	CA	94115	421-9963
Pintado	Jack	22 Hoover Ave.	Bowie	MD	20712	731-5134
Van Damme	Lucie	87 Main St.	Richmond	VA	23233	315-3545
Peppermint	Patty	127 Big St.	Lebanon	MO	92023	462-6267

Blabla	Blablabla	Bla
...	...	...

Flat and relational databases

A flat database usually consists of one table. In flat database management systems such as FileMaker Pro, the terms table and database are interchangeable.

A relational database presents a more sophisticated use of information. In relational database management systems, two or more tables are contained in the database. Therefore, you can store as many tables as you need in a single database file and each table could have one or more indexes. Tables can be linked so that information can be shared, saving you the trouble of copying the same information into several locations and eliminating the maintenance of duplicate information. Linking is important if there are relationships between the various pieces of information.

For example, if you want to add information to the entries contained in the address book in our first example, such as the company address and phone number, one way to do this would be to add them to the table:

Last Name	First Name	Address	City	State	Zip	Phone	Company	CoPhone
Jordan	Ann	6772 Toyon Court	San Mateo	CA	94403	349-5353	Rocco & Co.	526-2342
Brown	Charles	30 Saxony Ave.	San Francisco	CA	94115	421-9963	National Laundry	982-9400
Pintado	Jack	22 Hoover Ave.	Bowie	MD	20712	731-5134	Rocco & Co.	526-2342
Van Damme	Lucie	87 Main St.	Richmond	VA	23233	315-3545	Presto Clean	751-5290
Peppermint	Patty	127 Big St.	Lebanon	MO	92023	462-6267	Presto Clean	751-5290

However, adding this information might lead to duplication of information, given that some people might be working for the same company. To prevent duplication and to save on disk space and time required to update, you could create a new table containing only the business information. For example, the new table could be called: Companies. Each record of that new table would have a unique ID number, *Company Ref*, which would also be stored in the first table.

The database now contains two related tables, each having a field containing the common information, named “Company Ref”:

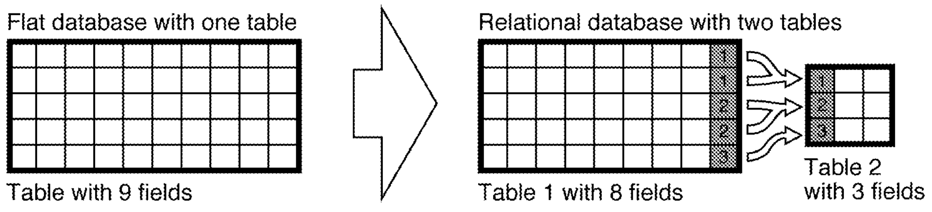
Table 1: containing information about each person:

Last Name	First Name	Address	City	State	Zip	Phone	**Company Ref**
Jordan	Ann	6772 Toyon Court	San Mateo	CA	94403	349-5353	**RO**
Brown	Charles	30 Saxony Ave.	San Francisco	CA	94115	421-9963	**NA**
Pintado	Jack	22 Hoover Ave.	Bowie	MD	20712	731-5134	**RO**
Van Damme	Lucie	87 Main St.	Richmond	VA	23233	315-3545	**PR**
Peppermint	Patty	127 Big St.	Lebanon	MO	92023	462-6267	**PR**

Table 2: containing information about the companies:

Company ref	Company	Phone
NA	National Laundry	982-9400
PR	Presto Cleaning	751-5290
RO	Rocco & Co.	526-2342

The two databases could also be compared as follows:



The relational database is smaller because it avoids useless data duplication. In order to retrieve full information about any given individual in your address book, you would perform a search in your first table, retrieve the company reference, and then perform a search in the second table. The flat model may be easier to manage when retrieving data given that only one search is required. However it tends to consume valuable disk space. The flat model also introduces the risk of data discrepancies due to the duplication of data.

Note: Relational Database Management Systems (RDBMS) are usually programmed with SQL (System Query Language) statements, which have the ability to automatically resolve relations between related tables.

Although INM V12 Database can store multiple tables per database, it relies on Lingo scripts to resolve relations. It cannot automatically resolve such relations.

Field types

For optimal data sorting and searching, you can specify the kind of information to be stored in each field. In INM V12 Database, fields can be designated to contain strings, integers, floating-point numbers, dates, pictures, sounds, palettes, etc. A field would then be of type **string**, **integer**, **float**, **date**, or **media**. Fields of type Media can accommodate any media that can be stored in a cast member. The known exceptions are Film Loops and QuickTime movies. See the section named Capacities and Limits for a formal definition of each field type.

For example, if you wanted to organize a contest where each person listed in your address book were collecting points, you would need to keep track of the number of points accumulated by each person. Therefore, you would update your address book to include a new field: *number of points*. Since you would want to search and sort this new field quickly, you would need to define an index. This new field could be one of two types: string or integer.

If you define the new field as type string, you might end up with the following listing when the table is sorted by ascending order of points:

Jordan	Ann	1
Brown	Charles	12
Peppermint	Patty	127
Pintado	Jack	6
Van Damme	Lucie	64

This order occurs because the string "12" is alphabetically lower than the string "6" given that the ASCII code for "1" is 49 which is smaller than the ASCII code for "6", 54. To sort the list in the expected ascending order, you must define the field *number of points* to be of type integer to get this result:

Jordan	Ann	1
Pintado	Jack	6
Brown	Charles	12
Van Damme	Lucie	64
Peppermint	Patty	127

Typecasting

Typecasting (or *casting*, for short) is the process of converting a piece of data from one type to another. This is a common mechanism to most programming languages, including Lingo.

For example, the integer *234* can be casted to the string "234". Conversely, the string "3.1416" can be casted to the float *3.1416*.

Typecasting can be performed explicitly in Lingo using the `Integer`, `String` and `Float` functions (i.e., `String(234)` returns the string "234") or automatically (i.e., `12&34` returns the string "1234").

INM V12 Database has the same ability as Lingo to typecast data when it is required by the context. However, some borderline conditions can lead to ambiguous results such as trying to store the value " 123" in a field of type `Integer` (note the leading space).

You must always make sure that the data supplied to INM V12 Database does not contain spurious characters, otherwise typecasting will not be performed properly.

International support

Although the 26 basic letters of the Roman alphabet sort in the same order in all roman languages, the position of accented characters (also called *mutated characters*) varies from one language to another. For example, the letter **ä** sorts as a regular **a** in German whereas it sorts after **z** in Swedish. Likewise, in Spanish, **ch** sorts after **cz** and **ll** sorts after **lz.**

Note: Everything that applies to the type `string` also applies to custom string types. Throughout this manual, the term `string` is used to designate both the default INM V12 Database string and custom string types.

INM V12 Database's default `string` was designed to satisfy as many languages as possible. It can sort and search texts in English, French, Italian, Dutch, German, Norwegian, etc.

INM V12 Database also offers the option of defining fields of type `Swedish`, `Spanish`, `Hebrew`, etc. that index and sort data in a way that is compliant with these languages.

The Regular Edition of INM V12 Database allows you to create custom string types, where you define a sort/search description table for each. Therefore, you can define your own string type for any language supported by single-byte characters, including Klingon.

For details on various character sets and custom string types, please see Appendix 3: String and Custom String Types.

Selection, current record, search criteria

Besides sorting a table through indexes, you can find information based on search criteria. You can define simple search criteria, also called **simple queries**, such as:

- First name is Jack
- State is California
- Number of points is less than 30
- Last name begins with P

Or you can define **complex search criteria**, also called Boolean queries using and/or, such as:

- First name is Jack **or** Last name begins with P
- State is California **and** Number of points is less than 30
- State is California **and** Number of points is less than 30 **and** Last name contains "pe"

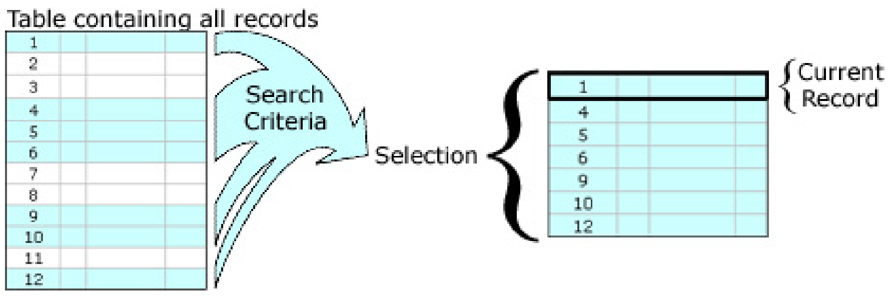
Note: Database Management Systems that use SQL as their programming language can define search criteria such as: *(Dish is soup or appetizer)* and *(Main Ingredient is celery or eggplant or pumpkin)*. INM V12 Database does not support alternating uses of ANDs and ORs.

See technical notes on http://www.INM.com/support/v12director/technotes for possible workarounds.

The **selection** is a set of records currently available in the table. When a table is opened the selection contains all the records of the table. If you search through a table after having defined search criteria, the resulting set of records that satisfy the search is the new selection. When a selection is first defined, the current record is the first record of that selection.

- If exactly one record satisfies the search criteria, the selection contains only that record, which automatically becomes the current record.
- If two or more records satisfy the search criteria, the selection is the set of those records, and the first record of the selection becomes the current record.
- If no record satisfies the search criteria, then the selection is empty and the current record is undefined. Any attempt to read or write in a field will result in an error.

This figure illustrates the idea of searching a table for records satisfying a certain criteria. The result is placed in a selection, the first record of which becomes the current record.



All operations on any fields (such as reading and writing data) are done on the current record. Therefore, before performing these operations, you must designate the record on which you wish to work as the current record by selecting it, and by using methods such as `mGoFirst`, `mGoLast`, `mGoNext`, `mGoPrevious` and `mGo`.

You can read the contents of a field in the current record, modify its contents or delete the entire record. When you move from one record to then next in the selection, the current record pointer changes. Note that if you modify the fields of the current record, you must call `mUpdateRecord` to save your changes to the database before moving to another record, otherwise your changes will be lost.

Using INM V12 Database- Step by Step

Overview

This section covers the main steps in using INM V12 Database. If you have looked at the *First Steps* tutorial (highly recommended), you should already be familiar with these five steps.

INM V12 Database basics

INM V12 Database is a powerful database management tool, composed of two Xtras libraries: a **database** Xtra named "V12dbe" and a **table** Xtra named "V12table". The database Xtra is used to create a new database or to open an existing database in a given mode (Read Only, ReadWrite or Create). The table Xtra is used to manage the content of the table in your database.

The main Ssteps

The *First Steps* tutorial described a typical step-by-step use of INM V12 Database. In this section those individual steps are explored in greater detail.

Step 1: Decide on a Data Model: Before you create your database, decide which fields are needed, the type of those fields, how they should be grouped in the tables and which fields should be indexed. This is a design effort that does not require a special tool (with the possible exception of a word processor to help you edit your ideas). If your original data is managed in FileMaker Pro, MS Access, or a similar database management product, that database's model is probably the best starting point for your V12 database model.

Step 2: Prepare the Data: If your original data is managed in FileMaker Pro, MS Access, or a similar database management product, in step 2, you make sure that your data is properly entered and that it is in a format readable by INM V12 Database (Text file, DBF file or one of INM V12 Database's ODBC-compliant formats).

Step 3: Create a Database: Use the INM V12 Database Editor to create the V12 database you designed at Step 1. Alternatively, you can use the database Xtra's (i.e. Xtra *V12dbe*'s) methods to write an automated database creation script in Lingo. (See the First Steps tutorial for sample code: http://www.INM.com/products/v12director/first-steps/)

Note: Although steps 1 and 2 do not involve any production work or programming, they are the **most critical** ones to the success of your project.

A well-designed project will yield high-quality results, on time, on budget. Similarly, failing to lay solid foundations at steps 1 and 2 will lead to an unmanageable project with fragile results. If you don't feel comfortable with steps 1 and 2, we recommend that you seek advice or hire professional help.
(See You're not alone!)

Step 4: Import Data into a V12 Database : Use the INM V12 Database Editor to import the Text or DBF file exported at Step 2. Alternatively, you can write Lingo scripts to automate the process of importing data into your INM V12 databases.

Step 5: Implementing the User Interface: In this step you develop the means by which users will search for, retrieve and modify data at runtime. This interaction with the database can be developed either using Behaviors attached to Director sprites, or as Lingo handlers in Director script members. Sample movies are available on Integration New Media's web site, at:
http://www.INM.com/support/v12director/technotes/. These movies each demonstrate specific techniques commonly used in database-driven applications and can be used to inspire the development of your projects.

Each of the aforementioned steps is discussed more in depth in subsequent sections. Since INM V12 Database offers more than one way to attain a goal, the simplest approach is explained first; then alternate and more powerful or versatile approaches are discussed.

Step 1: Decide on a Data Model

Before creating a database file, you need to decide how you want to organize your data. If your original data is managed in FileMaker Pro, MS Access, or a similar database management product, that database's model is probably the best starting point for your V12 database model. The questions you need to address are:

- which fields are required and what are their respective types?
- which fields should be indexed for quick searching and sorting?
- how many tables are required to group the fields?
- are there any relationships between the various tables?

In the stationary catalog example below, only one table is needed. It is called "Articles". The seven fields you need are :

- Field "ItemName" of type String
- Field "Category" of type String
- Field "Description" of type String
- Field "Price" of type Float
- Field "CatalogNumber" of type Integer
- Field "Photo" of type Media
- Field "Date" of type Date

Since only the fields "ItemName", "Price" and "CatalogNumber" will be searchable, only they are indexed.

Defining identifiers

Tables, fields and indexes are given names called **identifiers**, and INM V12 Database makes reference to them by use of these identifiers. An identifier must start with a low-ASCII alphabetic character (a..z, A..Z) and can be followed by any combination of alphanumeric characters (0..9, a..z, A..Z, à, é, ö, …). The maximum length for an identifier is 32 characters. No two fields or indexes of a table can have the same name.

INM V12 Database is not case-sensitive. That is, upper cases and lower cases are identical. These identifiers are considered identical in INM V12 Database: "articles", "ARTICLES", "Articles", "aRtICleS".

Step 2: Prepare the Data

Step 2 is relevant only if your original data is managed in FileMaker Pro, 4th Dimension, DBase or any other database management system that has the ability to export TEXT or DBF files.

If you plan to use an ODBC driver to import your data from MS Access, MS FoxPro, MS Excel or MS SQL Server, or if the records must be keyed-in by the user, skip to Step 3.

In brief, Step 2 consists in making sure that your original data is properly structured and in exporting it as Text or DBF files. Those files are then imported to V12 databases at Step 4: Import Data into a V12 Database .

TEXT file formats

Text files are the most popular data interchange file formats. Usually, *TAB-delimited* Text files are used to exchange data between database management systems.

A typical TAB-delimited file is in this format:

```
Field_A1 TAB Field_A2 TAB Field_A3 TAB ... TAB Field_An CR
Field_B1 TAB Field_B2 TAB Field_B3 TAB ... TAB Field_Bn CR
Field_C1 TAB Field_C2 TAB Field_C3 TAB ... TAB Field_Cn CR
```

where `Field_A1, Field_A2,` etc. represent the actual data in those fields. `TAB` is the ASCII character 9, indicating the end of a field.

On the Mac, `CR` is the ASCII character 13, indicating the end of a record. On Windows, `CR` is the ASCII character 13 followed by the ASCII character 10 (Line Feed). Since INM V12 Database always ignores Line Feed characters, you need not worry about exceptional cases between the Mac and Windows with respect to Record Delimiters.

Generally, using the INM V12 Database Editor or the `mImport` method to import a text file into a V12 Database file is a straightforward process, unless your fields contain `TAB` or `CR` characters. In such cases, INM V12 Database confuses the real delimiter with the legitimate content of your field. See Dealing with delimiter ambiguity below.

Field descriptors

INM V12 Database requires a special type of Delimited Text file format. The file's first line must contain **field descriptors**, or the names of the fields into which the data that follow must be imported. This file format is sometimes referred to as **mail merge format**. This is an example of such a file:

Name	*Price*	*CatNumber*
Ruler	1.99	1431
Labels	1.19	1743
Tags	6.19	...

You can easily have FileMaker Pro and MS Access export those field names before exporting the records data (See Exporting a MS Access Database to text).

Dealing with delimiter ambiguity

Most of the time, TABs are used to delimit fields in a Text file, and CRs to delimit records. If your fields contain TABs or CRs as part of their actual data, the legitimate content of your fields would be confused with those delimiters once exported in a text file. There is more than one way to deal with this problem. Choose the one — or combination — that best fit your project's needs in the list below.

Virtual carriage returns

Some database management systems (e.g., FileMaker Pro) export a special character other than ASCII #13 instead of the CRs that appear in your fields. For example, FileMaker Pro exports ASCII #11 (Vertical Tab) instead of ASCII #13. Those characters are called *Virtual Carriage Returns* or *VirtualCR* for short.

INM V12 Database can recognize those characters and convert them to real Carriage Returns (ASCII #13) once they are imported. See Step 4: Import Data into a V12 Database, Import data with mImport and VirtualCR in Properties of Databases.

Text qualifiers

A *text qualifier* is a special character used to begin and end each Text field. In most database management systems, the quotation mark (") is the default text qualifier. Its main purpose is to group a field's content between two identical marks so as to enable the occurrence of field and record delimiters without the risk of confusion.

Example:

"Name","Description" *CR*

"Hat","high-quality, excellent fabric, available in:*CR*Red*CR*Green*CR*Blue"

"Shoe","this description, field, contains, commas, and, Carriage*CR*Returns"

Text qualifiers are automatically placed in text files exported from MS Access, FileMaker Pro (Mail Merge format) and MS Excel (only for fields that contain commas).

Text files containing Text Qualifiers are easily imported to V12 databases by setting the `mImport` method's `TextQualifier` property to the right character. See Step 4: Import Data into a V12 Database, Import data with mImport.

Custom delimiters

Another way to avoid delimiter ambiguity is to choose delimiters other than TAB and CR. Some database management systems allow you to select appropriate delimiters before exporting to a TEXT file (e.g., 4th Dimension). Some others allow only the selection of a custom field delimiter and always use CRs as record delimiters (e.g., MS Access). FileMaker Pro and MS Excel do not allow for any customization.

INM V12 Database's `mImport` method assumes, by default, that the field and record delimiters are TAB and CR. However, other delimiters can be specified. See Step 4: Import Data into a V12 Database, Import data with mImport.

Note: Since INM V12 Database always ignores Line Feed characters (ASCII Character 10) they cannot be used as field or record delimiters.

Calculated fields

If your database management system does not support alternative delimiters you can nonetheless force it to export your own delimiters by creating an additional field and setting it as the result of the concatenation of all the other fields with the desired delimiter in between each two fields. Then, export only the new field in a text file.

Processing the exported text file

If the database management system used to store your data is not flexible enough, or if the data themselves are not properly structured, you can export them in a text file and use Third Party tools to search and replace sequences — or patterns — of characters.

Below is a non-exhaustive list of helpful tools:

- BBEdit from Bare Bones Software (http://www.BareBones.com/) For MacOS.
- TextPad from Helios Software Solutions (http://www.Textpad.com/). For Windows.
- UltraEdit from IDM Computer Solutions (http://www.Ultraedit.com/). For Windows.
- Microsoft Excel from Microsoft Corp. (http://www.Microsoft.com/. For MacOS and Windows.

BBEdit, TextPad and UltraEdit feature GREPs (General Regular Expression Parsers), which are very convenient to reorganize unstructured data.

Character Sets

Character sets are not standard across operating systems and file formats. For example, the letter "é" is the 233rd on Windows, whereas it is the 142nd on Macintosh and the 130th on MS-DOS.

Although all three operating systems use the ASCII characters set, only low-ASCII characters (i.e., those below #127) are common to the many variants of the ASCII set. Therefore, the rest of this topic is of interest to you only if you deal with high-ASCII characters (such as **€, å, æ, ß, ê, ï, ø, ž, ‰, §, ¥,** etc.)

Note: Most applications import/export DBF files using the MS-DOS character set.

INM V12 Database's `CharacterSet` property can be set to translate Windows, Macintosh or MS-DOS character sets when importing or exporting Text or DBF files. Optionally, `mImport` accepts the `CharacterSet` property to use only once to import a single file (as opposed to the `CharacterSet` property which permanently affects `mImport` and `mExportSelection`, or until it is set to another value).

MS Word documents, V12 databases as well as many other proprietary file formats are cross-platform compatible. You should not worry about this portability issue if your data contains only low-ASCII characters (e.g. English alphabet).

Dealing with dates

Although INM V12 Database can output dates in highly customizable formats, it requires that they be input in a single unambiguous format called the *raw* format: YYYY/MM/DD.

- YYYY: year in 4 digits (e.g., 1901, 1997, 2002)
- MM: month in 1 or 2 digits (e.g., 01 or 1 for January)
- DD: day in 1 or 2 digits (e.g., *04* or *4* for the 4th day of the month)

The separator between the three chunks of values can be any non-numeric character, although slash (`/`), hyphen (`-`) and period (`.`) are most commonly used.

Any date, that needs to be imported in a INM V12 Database field of type `date`, needs to be in this raw format. This rule applies to the INM V12 Database Editor as well as to INM V12 Database's Lingo methods that accept dates as input parameters (e.g., `mImport`, `mSetField` and `mSetCriteria`).

Note: If you fail to initialize a field of type Date in a new record, or try to store an invalid date in it, it is automatically set to 1900/01/01 (January 1st, 1900).

Exporting a FileMaker Pro database to text

In FileMaker Pro, choose File > Import/Export > Export Records and select "Merge (*.MER)" in the Save as Type menu. As a side effect, FileMaker Pro exports your data with quotation marks surrounding each field and a comma as field separator. Your file can easily be imported to the V12 database with quotation marks as Text Qualifiers (see Text qualifiers) and commas as field delimiters (see Custom delimiters).

Exporting a MS Access Database to text

In MS Access, choose File > Export, to an external file or database. Then, select Text Files in the Save as Type menu. Click Export or Save. Make sure that Delimited is selected and click Next. Choose an appropriate field delimiter for your data (see Dealing with delimiter ambiguity), choose a Text Qualifier from the list (see Text qualifiers), and check "Include Field Names in First Row"; then click Next.

Note: MS Access databases can be imported directly to V12 databases either by using the INM V12 Database Editor, or through Lingo. See Loading a descriptor from a source file.

DBF file formats

INM V12 Database can import DBF files two ways:

- on both MacOS and Windows, it can read DBF files of type Dbase III, Dbase IV, Dbase V, FoxPro 2.0, FoxPro 2.5, FoxPro 2.6, FoxPro 3.0 and FoxPro 5.0.
- on Windows only, DBF files can be imported using the FoxPro ODBC driver.

You may want to export your data as DBF files, if that format is supported by your database management system.

DBF is an old file format. It was enhanced over the years but most common applications still use the popular Dbase III format whose features are common to all other DBF file variants. Limitations include:

- Field names are limited to ten characters, all in upper case,
- The number of fields per DBF file is limited to 128,
- Records are of fixed length, determined upon the creation of the DBF file,
- There is more than one way to deal with high-ASCII characters (accented characters) with DBF files. This depends on the operating system and application used to manage the DBF file,
- Indexes are saved in separate files with extensions such as IDX, MDX, NDX or CDX (depends on the managing application),
- DBF files cannot be password-protected. However, some applications protect DBF files by encrypting/decrypting them,
- Character fields (roughly, the equivalent of INM V12 Database's string fields) are limited to 255 characters. Any text longer than 255 characters, must be stored in separate files called DBT files and referred to by Memo fields,
- Media (either Binary or Text) are stored in external DBT files pointed to by Memo fields in the DBF file. Media fields are limited to 32K of size. DBF fields of type Media are not supported by INM V12 Database.

Various flavors of the DBF file format were introduced over the years, such as DBase IV, DBase V, FoxPro 2.0, FoxPro 2.5, FoxPro 2.6, FoxPro 3.0 and FoxPro 5.0. They all include DBase III's features as core specifications and add new data types or extend certain limits. See Appendix 1: Database Creation and Data Importing Rules for more details.

In summary, the exact structure and limitations of your DBF files largely depend on how your database management system deals with them.

Note: Years ago, DBF files were convenient, given that they contained fewer variants than TEXT files. However, since the introduction of Windows and the popularization of DBF to other Operating Systems, DBF files now contain many categories and have become difficult to manage. INM V12 Database's preferred file importing format is Text.

Field buffer size

Prior to creating your database structures, you need to determine the size of the largest chunk of data for each field of type `string` or `media` in your database. This helps you optimize the size of the buffers needed to manage INM V12 Database's internal data structures for each of those fields.

If you are confident that your `string`s will not exceed 256 bytes, or your `media` 64K, you do not need to worry about the buffer size. Default buffers are set to 256 bytes for `string`s and to 64K for `media`.

> **Note:** Database management systems that use a fixed-length record format (such as the DBF file format) use this maximum value to allocate data space on disk. Consequently, that amount of space is lost for each record of the database regardless of the actual data stored in it.
>
> INM V12 Database uses a variable-length record format. This means that it uses the exact amount of space needed for the storage of a record on disk, with no space loss at all. The Field Buffer Size refers to the RAM buffer, used while transferring data between Director and the V12 database files.

Step 3: Create a Database

At Step 3, you formalize the database you designed at Step 1: Decide on a Data Model into a **database descriptor**. Then, you provide that descriptor to the INM V12 Database Editor (if you choose to use the INM V12 Database Editor), or to INM V12 Database's `mReadDBstructure` method (if you decided to script the database creation process).

The INM V12 Database Editor is a convenient point-and-click environment for small projects. Scripting the database creation process with Lingo requires a little more effort upfront but may end up saving you a lot of time, if you need to experiment with your database structure or data before committing to a final form. It enables you to automate the database creation process.

If you are going to use the INM V12 Database Editor, just read through the next two sections (Database descriptors and Using the INM V12 Database Editor) and skip to Step 4: Import Data into a V12 Database. If you wish to script the database creation process, read through Script the database creation as well.

Database descriptors

Following is the format of text (and literal) database descriptors required by both the INM V12 Database Editor and the `mReadDBstructure` method. It is used to build a database structure from scratch.

If you build your V12 databases from other databases (e.g., MS Access, MS Excel, etc.), you can directly skip to Using the INM V12 Database Editor or Script the database creation.

The desired V12 Database file structure is stored in a text file (or Director member) called the database descriptor in this format:

```
[TABLE]
NameOfTable
[FIELDS]
FieldName1 FieldType1 IndexType1
FieldName2 FieldType2 IndexType2
FieldName3 FieldType3 BufferSize3 IndexType3

(* if there are more than one table, their descriptors follow each
   other *)

[Table]
NameOfTable2
[FIELDS]
FieldName1 FieldType1 IndexType1
FieldName2 FieldType2 IndexType2
etc.

[END]
```

The `[TABLE]` tag is followed by one parameter: the name of the table. This is an identifier (see Defining identifiers).

The [FIELDS] tag is followed by as many lines as you need to define fields in the above defined table. The syntax of each line is as follows (see Database Basics for a thorough explanation of these concepts):

- FieldName: the name given to the field to be created. This is an identifier (see Defining identifiers),
- FieldType: string, integer, float, date, media or a custom string type (see Field typesField types),
- BufferSize: the amount of RAM to allocate for the internal management of the field's content. This parameter is relevant only for fields of type string and media. If omitted, fields of type string are created with a default buffer size of 255 characters and fields of type media are created with a default buffer size of 64K. See Field buffer size in Step 2: Prepare the Data,

  Note: If you try to store text longer than the size of the buffer allocated for the field type string, INM V12 Database signals a warning and stores the truncated text into the field. Media that are larger than the maximum buffer size of a field are not stored at all.

- IndexType: the word "indexed" if the field must be indexed, or the word "full-indexed" if the field must be full-indexed, or nothing if no indexing is required. If you need to both index and full-index a field, see Defining both an index and a full-index on a field.

The [END] tag indicates the end of the descriptor. It is a mandatory tag.

In each line of the descriptor file, tokens (i.e. field name, index name, value, etc.) must be separated by one or multiple Tabs and/or Space characters.

Example:

```
[TABLE]
Recipes
[FIELDS]
NameOfRecipe string indexed
Calories integer indexed
CookingTime integer
TextOfRecipe string 5000 full-indexed
Photo media 300000
[END]
```

Note: A convenient way to build a Descriptor File for a database containing a large number of tables, fields or indexes is to type it into a spreadsheet thus taking advantage of advanced editing functions. The results can then be saved into a TAB-delimited file or Copied and Pasted into a Director field for processing by mReadDBStructure.

Note: A valid database needs at least one table, and each table requires at least one field and at least one index.

Defining both an index and a full-index on a field

In exceptional cases, you would need to define both an index and a full-index on a field. Since the IndexType parameter defined above can represent only one of "indexed" or "full-indexed", you would need to set it to "indexed" and define the full-index separately under an additional tag named [FULL-INDEXES].

The [FULL-INDEXES] tag must follow the [FIELDS] section and must be followed by a list of fields to be full-indexed, one per line.

Example:

```
[TABLE]
Recipes
[FIELDS]
NameOfRecipe string indexed
Calories integer indexed
CookingTime integer
TextOfRecipe string 5000 indexed
Photo media 300000
[FULL-INDEXES]
TextOfRecipe
[END]
```

Alternate syntax for creating indexes

Database descriptors support an alternate syntax for the creation of indexes. The [INDEXES] tag can be used right after the field definitions to explicitly name and define the desired indexes.

This alternate syntax is used by mDumpStructure for clarity (see View the structure of a databaseView the structure of a database). It also allows the definition of unique-valued indexes and descending indexes which are used only in exceptional cases (in summary, if you don't know what they mean, you probably don't need them).

This database descriptor example is equivalent to the one above:

```
[TABLE]
Recipes
[FIELDS]
NameOfRecipe string
Calories integer
CookingTime integer
TextOfRecipe string 5000
Photo media 300000
[INDEXES]
NameOfRecipeNdx duplicate NameOfRecipe ascending
CaloriesNdx duplicate Calories ascending
TextOfRecipeNdx duplicate TextOfRecipe ascending
[FULL-INDEXES]
TextOfRecipe
[END]
```

Compound indexes

Compound indexes are indexes defined on two or more fields (see Database Basics / Compound indexes). Compound indexes can be defined after the [INDEXES] tag, as in:

```
[TABLE]
Students
[FIELDS]
LastName string
FirstName string
Age integer
[INDEXES]
CompoundNdx duplicate LastName ascending FirstName ascending
[END]
```

The general syntax of a compound index definition is

```
[INDEXES]
Indx1 UniqueOrDup [FieldName AscOrDesc]1..10
```

where:

- Indx1 is the name of the compound index
- UniqueOrDup is either "unique" or "duplicate", depending upon whether or not you allow duplicate entries for that index
- FieldName is the name of a field defined under the [FIELDS] tag
- AscOrDesc is "ascending" if you want that field sorted low-to-high, or "descending" otherwise.

Up to ten FieldName AscOrDesc couples can be defined for a single compound index.

Adding comments to database descriptors

Database descriptors can also contain comments in much the same way Lingo scripts do. In Lingo, comments are preceded by double hyphens ("--") and must be followed by a CARRIAGE_RETURN. In database descriptors, comments must be preceded by (* and be followed by *). They can include any sequences of characters, including CARRIAGE_RETURNs.

Example:

```
(*
   description of the Mega-Cookbook recipes table version 1.1
   by Bill Gatezky, 14-Feb-99
  This is a valid comment despite the fact that it contains
  Carriage Returns
*)
[TABLE]
Recipes
(* this is also a valid comment *)
[FIELDS]
NameOfRecipe string indexed
...
[END]
```

The comment opening tag for database descriptors must be followed by a blank character such as a space, tab or CARRIAGE_RETURN. Likewise, a comment closing tag must be preceded by a blank character. Thus,

```
(*invalid comment: will generate an error*)
```

is an invalid comment, whereas

```
(* valid comment *)
```

is valid.

Multiple tables in a descriptor

If your database has more than one table, each new table follows the description for the previous table, before the [END] tag.

Example:

```
[TABLE]
NameOfTable1
[FIELDS]
FieldName1 FieldType1 IndexType1
FieldName2 FieldType2 IndexType2
FieldName3 FieldType3 BufferSize3
[Full-INDEXES]
FieldName3Ndx duplicate ascending

[TABLE]
NameOfTable2
[FIELDS]
FieldName1 FieldType1 IndexType1
FieldName2 FieldType2 IndexType2

etc.

[END]
```

Using the INM V12 Database Editor

To create a V12 database using the INM V12 Database Editor:

1. Choose File > New...
2. Fill out the Database Descriptor field according to the syntax described in Database descriptors, or load a descriptor from an existing file (see Loading a descriptor from a source file),
3. Provide a name, and optionally a password, for your new V12 database,
4. Click the Create button

Loading a descriptor from a source file

Instead of filling out the Database Descriptor field manually in the INM V12 Database Editor, you can load one from an external file. Click the **Source** list to select the type of the file that contains the descriptor information. File types that are supported by the INM V12 Database Editor include:

- Text
- DBF file
- V12 file
- Template
- FoxPro file
- Access file
- Excel file
- SQL Server Database

If your data is already in one of the database file formats listed here, you can simply use that database to retrieve the descriptor information. Click the **Load…** button and browse to the file that contains the descriptor information.

Once the descriptor information appears in the Descriptor box, you may edit it. For instance, your source database may contain several tables, and you may only want to include a subset of these in your V12 database.

For more information, see the user manual for the INM V12 Database Editor. The INM V12 Database Editor and User Manual are available for download from the Free Tools section of Integration New Media's web site.

http://www.INM.com/products/v12director/tools/

Script the database creation

Automating the creation a V12 Database file through Lingo with INM V12 Database consists of three steps:

- Create an Xtra instance of the database with `New`
- Define its structure with `mReadDBstructure`
- Build the database with `mBuild`

The general form of a database creation Lingo handler is:

```
on CreateDatabase
   gDB = New(Xtra "V12dbe", FileName, "create", Password)
   CheckV12Error()
   mReadDBStructure(gDB, InputType, other params)
   CheckV12Error()
   mBuild(gDB)
   CheckV12Error()
    gDB.mClose()
   gDB=0
end CreateDatabase
```

where:

- `FileName` is the full pathname of the V12 database to create
- `Password` is the password to protect `FileName`
- `InputType` is one of "Text", "Literal", "DBF", "V12", "FoxPro", "Access", "Excel" or "SQL Server".

- *other params* are one or more parameters depending on the selected InputType.
- The resulting INM V12 Database file can be immediately verified with mDumpStructure (see View the structure of a database).

mReadDBStructure reads the *structure* of a database file, not its content. To import the content of a database file, see Appendix 1: Database Creation and Data Importing Rules.

See CheckV12Error in Errors and Defensive Programming for a definition of the CheckV12Error() handler used throughout this section.

Step 3a: Create a database Xtra instance

Use the New method to create a database Xtra instance.

Syntax:

```
gDB = New(Xtra "V12dbe", Name, "create", Password)
```

The parameters you provide are:

- Name: the name of the new database file, including its path if needed (see Dealing with pathnames in Using Xtras).
- "Create" or the Mode: the mode in which the Xtra instance is defined. In this case, the mode is Create (create a new database file). Other possible modes are ReadOnly and ReadWrite. See Open an existing database.
- Password: the password is required if you wish to protect your database against tampering and/or data theft. You can lock the database with a password, but make sure to record it in a safe place. If you forget it, you will not be able to open your database again.

Example:

```
gDB = New(Xtra "V12dbe", "Catalog.V12", "Create", "top secret")
```

Note: For a number of reasons, the creation of an Xtra instance can fail (insufficient memory, invalid file path, etc.) Always make sure that your database instance is valid by checking V12Error (see CheckV12Error) or ObjectP (see Checking if New was successful in Using Xtras) before pursuing the database creation process.

Step 3b: Define the database structure

The next method, after successfully creating a database Xtra instance, is to call mReadDBstructure to read in the database structure you designed at Step 1: Decide on a Data Model.

mReadDBstructure requires one the following inputs:

- a database descriptor as defined in Database descriptors. Such a descriptor is supplied either as a text file or as a literal (i.e. a Director field or variable),
- a DBF file (DBase) which serves as a table template,
- a V12 database which serves as a database template,

- a directory containing one or more MS FoxPro files which serve collectively as a database template (Windows only, requires the FoxPro ODBC driver),
- a MS Access database which serves as a database template (Windows only, requires the Access ODBC driver),
- a MS Excel workbook which serves as a database template (Windows only, requires the Excel ODBC driver),
- an MS SQL Server data source, which serves as a database template (Windows only, requires the MS SQL Server ODBC driver).

See Appendix 1: Database Creation and Data Importing Rules for complete examples of each of the above variations of `mReadDBStructure`.

Note: A valid database needs at least one table, and each table must contain at least one field and at least one index.

Note: `mReadDBStructure` reads the *structure* of a DBF file, not its content. Use `mImport` to import the content of a database file.

It is always a good practice to check the value returned by `V12Error()` or `V12Status()` after calling `mReadDBstructure` (see Errors and Defensive Programming) to find out if an error occurred. You may also call `mDumpStructure` right after calling `mReadDBstructure` to check the actual database structure INM V12 Database will build once `mBuild` is called.

Database structure translation rules from the above ODBC-compliant databases to V12 Databases vary according to the specific ODBC driver installed on your computer.

Step 3c: Build the database

Once the database structure is read by `mReadDBstructure`, whether from a text file, a DBF file or otherwise, build the database by calling `mBuild`. `mBuild` checks if the database is well defined and creates the file on your disk.

Syntax:

```
mBuild(gDB)
```

`mBuild` optionally accepts a second parameter, "online", that makes the created file compatible with the INM V12 Database Online companion. In this case, two additional fields, named `_uID` and `_timeStamp` are created for INM V12 Database Online to manage internally. Both fields are hidden and do not appear in `mDumpStructure`'s result.

Note: For `mBuild` to create a licensed database (that is, one that does not display a Demo dialog when opened), a INM V12 Database license file must be present on your Mac or PC. Since the INM V12 Database license file cannot be delivered to the end-user, `mBuild` cannot be used to create new databases at runtime. If your application needs to create new databases at runtime, use `mCloneDatabase`
(see Cloning a Database).

Syntax:

```
mBuild(gDB, "online")
```

Example:

```
mBuild(gDB)
-- since mBuild does a lot of validations, checking
-- for errors/warnings is HIGHLY recommended
if V12Status() then
   Alert "mBuild failed with error code" & V12Error()
end if
```

Once the database file is built, the database instance remains valid and data can be immediately imported into the file. It is as if the database was opened in ReadWrite mode.

View the structure of a database

You can view the structure of a database with `mDumpStructure`.

Syntax:

```
mDumpStructure(gDB)
```

Example:

```
put mDumpStructure(gDB) into field "myDBstructure"
```

The above example places the structure of the database referred by `gDB` in the member named "myDBstructure".

```
(*
    Structure of file 'HardDisk:myDatabase.V12'
    created on Thu Apr 04 15:55:07 2020,
    last modified on Tue Apr 09 15:31:53 2020,
    file format version = V12,3.3.0,Multi-User
*)

[TABLE]
Articles

[FIELDS]
name string 256
category string 256
price Float
catalognumber Integer
description string 600

[INDEXES]
nameNdx duplicate name ascending     (* Default index *)
categoryNdx duplicate category ascending
priceNdx duplicate price ascending
cat#Ndx unique catalognumber ascending
catNameNdx duplicate category ascending name descending

[FULL-INDEXES]
description

[END]"
```

Note that the date/hour of the last modification mentioned in the header of the above output is provided by the Operating System. Therefore, it reflects the date/hour at which the V12 database was *closed* regardless of when the *modification* occurred.

This output is fully compatible with the database descriptors discussed in Database descriptors and thus, can be used as is with `mReadDBstructure`.

Step 4: Import Data into a V12 Database

In Step 3: Create a Database, you created a properly structured (although empty) V12 database. Step 4 explains how to import the data prepared at Step 2: Prepare the Data into your V12 database.

You can import data into a V12 database through one of the two following methods:

- using the INM V12 Database Editor. This is a convenient point-and-click environment for small projects.
- using INM V12 Database's `mImport` method in a Lingo handler. This approach is efficient when you need to experiment with your database structure or data before committing to a final form. However, it requires a bit more up-front effort to write/adapt Lingo handlers than simply using the INM V12 Database Editor.
- For extensive examples on how to import databases from a variety of sources, including Microsoft Access, Microsoft Excel, MS SQL, FoxPro, DBF and Text, see Appendix 1: Database Creation and Data Importing Rules.

Import data with the INM V12 Database Editor

To import data using the INM V12 Database Editor:

1 Choose File > Open... to open the V12 database you want to import data to. A newly created V12 database automatically opens and data can be immediately imported to it.

2 In the File menu, you will see the following options for importing data:

- Import Text File...
- Import DBF File...
- Import from V12...
- Import from FoxPro...
- Import from Excel...
- Import from Access...
- Import from SQL Server...

Choose the appropriate option for the format for your data.

3 Browse through your disk to locate the file containing the data to import and fill in any other information necessary to open the file. For some formats you may also need to specify a table name. Click Import.

If the source data is in more than one file, you can successively import them by repeating the above steps.

Script the data importing

mImport imports data to a INM V12 Database table both at authoring time (i.e., in Director's development environment) and at runtime (i.e., from a Projector or Shockwave movie).

mImport is very flexible and can be adapted to a large number of situations. It can import data from:

- a Text file
- a literal value, such as a string, a Director member, etc.
- a DBF file
- a V12 database
- a Lingo list or Lingo property list
- an MS Access database through an ODBC driver (Windows only)
- a FoxPro file through an ODBC driver (Windows only)
- an MS Excel file through an ODBC driver (Windows only)
- an MS SQL data source through an ODBC driver (Windows only)

Data type translation rules from the above ODBC-compliant databases to V12 Databases vary according to the specific ODBC driver installed on your computer.

The general form of a table importing script is:

```
-- create a V12dbe instance
gDB = New(Xtra"V12dbe", database_filename, mode, password)
CheckV12Error()

-- create a V12table instance
gTable = New(Xtra "V12table", mGetRef(gDB), TableName)
CheckV12Error()

-- import data
mImport(gTable, InputType, InputSource, other params)
CheckV12Error()

-- free the V12table and V12dbe instances
gTable.mClose()
gTable = 0
gDB.mClose()
gDB = 0
```

As with any V12table method, valid instances of V12dbe and V12table must exist before the method is invoked. This is explained in detail in Creating an instance of the INM V12 Database Xtra.

mImport's syntax varies significantly according to the selected input source. This is explained in details in Import data with mImport below.

Setting Xtra instances to *0* when they are no longer needed is mandatory, as explained in Closing the Xtra Instance, so to make sure that the imported data is secured on hard disk.

CheckV12Error is a generic error management handler explained in Errors and Defensive Programming.

Import data with mImport

The general syntax for `mImport` is:

```
mImport(gTable, InputType, InputSource, other params)
```

where:

- `InputType` is one of "Text", "DBF", "literal", "list", "propertyList", "V12", "Access", "FoxPro", "Excel" or "SQLserver".
- `InputSource` is the data to import or a reference to the data to import. It varies according to the selected `InputType`.
- *`other params`* are parameters that depend upon the selected `InputType`. For example, if `InputType` is "text", *`other params`* is an optional property list that specifies the source text file's field delimiter, record delimiter, etc. If `InputType` is "Access", *`other params`* are the user name, password and table to import. The details are explained below.

See Appendix 1: Database Creation and Data Importing Rules for complete examples of each of the above variations of `mImport`.

Using media with V12 Databases

Although INM V12 Database files can store different types of media there are alternatives to storing media directly in the database.

Instead of storing media in INM V12 Database files, they can be stored in Director cast members or as external files, which are assigned to members at run time.

In addition, these members' names or numbers can be stored in INM V12 Database tables to maintain their relationships to other data.

Advantages to referencing external media

- If your original media are already located in Director members, keeping just a reference to the media in your V12 database yields faster access times given that it avoids unnecessary memory allocations and re-allocations in transferring data between Director and INM V12 Database.
- If you need to make changes to your media, you don’t have to do any scripting to reimport the changed elements into your V12 database.
- If your media files are large (over 50Kb each), it takes longer to each edit record if the media is stored within the database. Note that the maximum size allowed for a field of type Media is 1Mb.

When to store media in a V12 Database

Sometimes storing media directly in INM V12 Database files has its advantages.

- Your data becomes completely independent of your Director projector and it may be easier to update.
- Bitmaps that are small, such as thumbnail images, can be stored in the database with no adverse effects on speed.
- A V12 database can be password protected; therefore everything that is stored in the database is secure from accidental or illegal modification. Note that this protection might be insufficient for sensitive or confidential data.

Examples of embedded and linked media

The NetCat demo, which can be downloaded from:

http://www.INM.com/products/v12director/demos/
is available in two versions, one with images embedded in the database and the other with links to the images. Through this demo you can see for yourself the differences in space requirements and performance.

The First Steps tutorial shows an example of how to store a reference to a media file within a V12 database, and use that field to insert media into your project. Download the First Steps tutorial from Integration New Media's web site:

http://www.INM.com/products/v12director/first-steps/

There is also a detailed example of how to use the INM V12 Database Lingo methods to import media into a V12 database in Appendix 1: Database Creation and Data Importing Rules/ Importing Media into a V12 Database.

Step 5: Implementing the User Interface

Steps 1 through 4 (Step 1: Decide on a Data Model through Step 4: Import Data into a V12 Database) explain how to design, build and import data into a V12 Database file. This section discusses the elements needed to manage your INM V12 Database files at runtime.

There are two basic strategies you can follow when creating your Macromedia Director front-end to your data:

1 Using the INM V12 Database Behaviors Library

2 Using Lingo

Tip: If you chose to script the database creation and importing processes, once the database file is ready, you do not need those Lingo scripts any longer. Moreover, they do not necessarily need to be delivered to the end-user.

At this point, you may want to consider removing those scripts, or storing them in an appropriate place. You may also want to keep all the scripts related to database creation in a separate Director movie.

Using the INM V12 Database Behaviors library

The fastest and easiest way to implement INM V12 Database into your project's user interface is to use the INM V12 Database Behaviors Library. See the *First Steps* manual and the *INM V12 Database Behaviors Library* manual for an overview of INM V12 Database Behaviors.

The V12 Behaviors Library is a free set of tools that allows you to quickly implement a user interface to the data, but is not as flexible as the Lingo approach. It may not have all of the functionality that you may require for your data model, or may not be able to present the data to the end-user the way you would like. The Behaviors allow you to implement the most commonly requested features, but at the sacrifice of high-end features and flexibility. Before you begin using the INM V12 Database Behaviors, analyze your needs and determine the functionalities your project requires. Once you start to implement your user interface with the standard V12 Behaviors, if there are a few features missing, you may find that it takes more work to modify the Behaviors to add the functionality you need, than to use the Lingo-based approach from the beginning.

The INM V12 Database Behaviors Library contains the following Action Behaviors (see the separate *INM V12 Database Behaviors Manual* for detailed information on these Behaviors and how to implement them in your project):

- Open Database

- Close Database
- Browse
- Go to Record
- Search All
- Search with one criterion
- Search with two criteria
- Search with three criteria
- Add Record
- Delete Record
- Delete Selection
- Import DBF File
- Export DBF File
- Import INM V12 Database File
- Import Text File
- Export Text File

If your project's user interface needs are met by the above list, then you may benefit from using the INM V12 Database Behavior Library, but if you need to implement features that aren't on this list, you may want to investigate further.

For instance, if your user interface requires that your users be able to specify nine search criteria for a search, then the Behaviors will not do what you want. Your only recourse at this stage would be to modify the Behaviors (requiring an advanced knowledge of Lingo), or to start using the Lingo approach. We recommend that you carefully investigate both options before committing to one or the other. You may find that Behaviors save you a lot of time in implementing user interfaces for your projects. You can also use Behaviors and custom Lingo scripts in the same project.

You can download the INM V12 Database Behavior Library and User Manual from the Free Tools section of Integration New Media's web site:

http://www.INM.com/products/v12director/tools/

The Lingo methods are much more flexible and powerful, but they *may* also take longer to implement in the beginning. The Lingo methods expose every feature of V12 to your projects, and will allow you to implement every data model and feature that V12 supports.

Tip: Before you commit yourself to using the INM V12 Database Behaviors Library in your project, you may want to ask Support@INM.com or other INM V12 Database users on V12-L <http://www.INM.com/support/list/> for advice.

Using Lingo

As for any INM V12 Database method, a valid V12dbe or V12table Xtra instance (depending on which Xtra the method belongs to) must exist before the method is invoked.

Generally, you create instances of V12dbe and V12table `on StartMovie`, store their references in global variables and use those instances throughout your project.

Likewise, `on StopMovie`, you set those global variables to 0 thus disposing of the Xtra instances and closing the V12 database file.

The creation of such Xtra instances is often referred to as Opening a Database and Opening a Table. Disposing of the Xtra instances is often referred to as Closing the Database and Closing the Table instances.

Open and close a database, a table

Open an existing database

Use the `New(Xtra "V12dbe"…)` method to open an existing V12 database. If your V12 database is not created yet, see Step 3: Create a Database to learn how to create it.

Syntax:

```
gDB = New(Xtra "V12dbe", database_filename, mode, password)
```

Opening a database means creating a `V12dbe` Xtra instance with the following parameters:

- `database_Filename`: the `name` of the database file. This is usually a filename preceded by the Lingo function `the MoviePath &` to indicate that the file is located in the same folder as the current movie (see Dealing with pathnames).
- `mode`: the mode in which the Xtra instance is opened. To allow for modifications to the database, open it in "ReadWrite" mode. If you open it in "ReadWrite" mode, only one user at a time can access your database. If you do not allow modifications to your database, open it in "ReadOnly" mode.
- `password`: the password. If you do not use the correct password, the database cannot be opened.

Example:

```
gDB = New(Xtra "V12dbe", the MoviePath & "Catalog.V12", "ReadWrite",
  "top secret")
```

Always make sure that the `New` method has succeeded by checking the validity of the returned reference with `ObjectP`. Example:

```
gDB = New(Xtra"V12dbe", the MoviePath & "Catalog.V12", "ReadWrite",
  "top secret")
if NOT (ObjectP(gDB)) then alert "New V12dbe failed"
```

Open a table

Records belong to tables. Creating new records, reading the contents of records, and searching and sorting records are all operations that are performed on tables. Prior to performing any of these operations, you must create a table Xtra instance.

Syntax:

```
gTable = New(Xtra "V12table", mGetRef(gDB), TableName)
```

To create a table Xtra instance, use the New method with the following parameters:

- gDB: the database Xtra instance to which the current table belongs
- TableName: the name of the table to open

Example:

```
gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")
```

mGetRef is a standard Xtra method that returns the exact reference of an Xtra instance.

Always make sure that the New method has succeeded by checking the validity of the returned reference with ObjectP.

Example:

```
gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")
if NOT (ObjectP(gTable)) then alert "New V12table failed"
```

This is a complete example of a script that would run on StartMovie:

```
on StartMovie
   global gDB, gTable
   gDB = New(Xtra "V12dbe", the MoviePath&"Catalog.V12", "ReadWrite",
   "pwd")
   CheckV12Error()
   gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")
   CheckV12Error()
   -- other init instructions
end StartMovie
```

Close a table

To close a table, first call the mClose() method with the table variable as a parameter. Then set the table reference variable to 0.
Example:

```
gTable.mClose()
gTable = 0
```

Close a database

To close a V12 database, first call the mClose() method with the database variable that refers to it as a parameter. Then set the database reference variable to 0.
Example:

```
gDB.mClose()
gDB = 0
```

Always make sure to dispose of all V12table instances before you dispose of the V12dbe instance that contains them.

This is a complete example of a script that would run on StopMovie:

```
on StopMovie
   global gDB, gTable
    gTable.mClose()
   gTable = 0
    gDB.mClose()
   gDB = 0
end StopMovie
```

Selection and current record

To read or write data to a record, set it as the current **record**. The current record concept is strongly related to the concept of **selection**. Both concepts are fundamental to this section. See Database Basics earlier in this manual for more details.

At any time, the selection is sorted according to one of its fields. You can enforce that sorting order with `mOrderBy` (see Sort a selection (mOrderBy)). Otherwise, the selection's sorting order would be defined by the index chosen by INM V12 Database for its last search. The field that determines the selection's sorting order is called the **master field**.

Selection at startup

When a table is first opened, its selection is the entire content of that table sorted by the field that is indexed by the default index. The first record of that selection – which is also the first record of the table – *is* the current record. The default index is the first index that was defined for the table in the database descriptor. You can use `mDumpStructure` to verify which of the table's indexes is the default index (see View the structure of a database).

You never need to explicitly manage indexes in INM V12 Database. The best index is always chosen by INM V12 Database to perform a search.

Select all the records of a table

Call `mSelectAll` at any time to set the selection to the whole table.

Syntax:

```
mSelectAll(gTable)
```

Example:

```
mOrderBy(gTable, "price", "ascending")
mSelectAll(gTable)
```

This example sets the selection to the whole table as referred to by `gTable`, in ascending order of prices (least to most expensive). The field "price" must be indexed for `mSelectAll` to work efficiently. Otherwise, it would be very slow.

Note: If you want the results of your query to be sorted, always call `mOrderBy` before calling `mSelectAll`. See Sort a selection (mOrderBy)

Browse a selection

Browsing a selection means changing the position of the current record. The following methods enable you to change the current record in a selection (to set the current record to various values related to a given selection).

mGetPosition

`mGetPosition`checks the position of the current record in a table and returns an integer between one and the total number of records in the selection.

Example:

```
        currRec = mGetPosition(gTable)
        put "the current record is:" & currRec
        -- returns the current record's position in the Message Window.
```

mGoNext

`mGoNext` sets the current record to the record following the current record.

Example:

```
        mGoNext(gTable)
```

Suppose that the current record is the tenth item in the selection. After calling `mGoNext`, the current record becomes the eleventh. If the selection contains only ten records, the current record does not change and a warning is reported by INM V12 Database (see Errors and Defensive Programming).

mGoPrevious

`mGoPrevious` sets the current record to the record preceding the current record.

Example:

```
        mGoPrevious(gTable)
```

Suppose that the current record is the tenth item in the selection. Upon calling `mGoPrevious,` the current record becomes the ninth. If the current record is the first record of the selection, upon calling `mGoPrevious` the current record does not change and a warning is reported by INM V12 Database (see Errors and Defensive Programming).

mGoFirst

`mGoFirst` sets the current record to the first record of the selection.

Example:

```
        mGoFirst(gTable)
```

mGoLast

`mGoLast`sets the current record to the last record of the selection.

Example:

```
        mGoLast(gTable)
```

mGo

`mGo` takes an integer parameter (call it *n*) and sets the current record to the *n*th item of the selection.

Example:

```
        mGo(gTable, 11)
```

This example sets the current record to the eleventh record of the selection. If no such record exists, `mGo` signals a warning.

mFind

`mFind` sets the current record to one, in the selection, whose Master Field equals or starts with the keyword provided as a parameter (see definition of Master Field in Selection and current record).

mFind is a great complement to mGo, which can set the current record only based on its position in the selection.

The syntax is:

```
mFind(gTable, "First", Keyword)
mFind(gTable, "Next")
mFind(gTable, "Previous")
```

where Keyword is the value to look for in the Master Field. If the Master Field is of type String, the matching record's content must *start with* Keyword. If it is of type Integer, Float or Date, it must *equal* Keyword.

Use the first form (with the "First" parameter), if you want the new current record to be the first one of the selection that matches Keyword

Use the second form (with the "Next" parameter) if you want it to be the next record in the selection relative to the present current record. Use the third form ("Previous") if you want it to be the previous record in the selection relative to the present current record.

Note: Because mFind uses the selection's Master Field, it is advised that you call mOrderBy with the appropriate field before calling mSelect and mFind. (see Search data with mSetCriteria).
If you don't call mOrderBy, mFind sets the current record based on the Master Field chosen by default by INM V12 Database, which is either the one indexed by the default index (if the table was just opened), or the one indexed by the best index chosen by INM V12 Database during the last search.

If, for example, you run this script:

```
mSetCriteria(gTable, "Age", ">", 30)
mOrderBy(gTable, "LastName")
mSelect(gTable)
```

and get this selection:

FirstName	LastName	Age
Marie	Curie	39
Albert	Einstein	75
Kurt	Gödel	36
Mona	Karp	53
Joe	Karp	31
Richard	Karp	62
Eric	Kartman	31
Marshall	McLuhan	48
Claude	Shannon	33
Alan	Turing	36
John	Von Neumann	51

The selection's Master Field is "LastName". Thus, a call to mFind would automatically look for values in this field. For example:

```
mFind(gTable, "First", "Kar") -- current rec becomes Mona Karp's
mFind(gTable, "Next") -- current rec becomes Joe Karp's
mFind(gTable, "Next") -- current rec becomes Richard Karp's
mFind(gTable, "Next") -- current rec becomes Eric Kartman's
mFind(gTable, "Next") -- current rec remains Eric Kartman's
mFind(gTable, "Previous") -- current rec becomes Richard Karp's
```

mFind can be used to quickly locate one occurrence of a keyword in a selection where many duplicate values exist, as opposed to mSetCriteria and mSelect, which find all occurrences but need more time.

Read data from a database

In order to read or write the content of a record, you must first set it as the current record. Setting the appropriate current record is accomplished by use of the mGoNext, mGoPrevious, mGoFirst, mGoLast, mGo and mFind methods (see Browse a selection).

Read Fields of Type String, Integer, Float and Date

Once the current record is properly set, mGetField retrieves the data from a specific field. mGetField retrieves data from all field types except Media.

Syntax:

```
var = mGetField(gTable, fieldName[, dataFormat])
```

Example:

```
cost = mGetField(gTable, "price")
```

This example stores the content of the price field from the current record in the variable cost. You do not need to specify the type of field you are reading. The Lingo variable is automatically set to the appropriate type after a successful call to mGetField (see Typecasting in Database Basics).

Example:

```
cost = mGetField(gTable, "price", "9,999.99")
```

This example retrieves the formatted content of the price field to the cost variable. The formatting is according to the pattern "9,999.99". That is, if the field price contains the value 1245.5, the string "1,245.50" is returned by mGetField. Note that the result of a formatted value is always a string.

Data formatting applies to mGetField the same way it does to mDataFormat. If two distinct formatting patterns are applied to a field with the mGetField option and mDataFormat, the mGetField option overrides mDataFormat. See Data formatting for a complete explanation on formatting patterns.

Note: mGetField retrieves only unformatted text. If you store styled text in a V12 record, you can retrieve the text without the style formatting using mGetField and you can retrieve the styled text with mGetMedia
(See Styled text.)

Read one or more entire records

mGetSelection allows for the retrieval of one or more fields in one or more records of the selection. The result type is one of the following:

- "Literal" - a string where fields are delimited by TABs and records by CARRIAGE_RETURNs (the default delimiters), or by any other custom delimiters you specify.
- "List" - a Lingo list of lists, where each sub-list represents a record and each item of each sub-list is the data contained in the corresponding field

- "PropertyList" - a Lingo list of property lists, where each sub-list represents a record and each item is a property/value pair: the property is the name of the field and the value is the data contained in it.

mGetSelection is powerful and flexible. Its behavior depends on the syntax used to call it. The syntax for mGetSelection to return a result of type String is:

```
mGetSelection(gTable, "Literal", [From [, #recs [, FieldDelimiter [,
  RecordDelimiter [, FieldNames ]* ]]]])
```

The syntax for mGetSelection to return a Lingo list is:

```
mGetSelection(gTable, "List" [, From [, #recs [, FieldNames ]* ]])
```

The syntax for mGetSelection to return a Lingo property list is:

```
mGetSelection(gTable, "PropertyList" [, From [, #recs [, FieldNames
  ]* ]])
```

where:

- gTable is the instance of the table from which records must be retrieved (mandatory parameter),
- From is the number of the first record to retrieve data from. It is optional. The default value is 1.
- #recs is the number of records to retrieve starting from record number From. It is optional. The default value is the number of records between From and the end of the selection plus 1 (convenient to retrieve all the records of a selection starting from record number From).
- FieldDelimiter is the character to use as the field delimiter. It is optional. The default field delimiter is a TAB.
- RecordDelimiter is the character to use as the record delimiter. It is optional. The default field delimiter is a CARRIAGE_RETURN.
- FieldNames are the names of the fields to retrieve, in the specified order. If the field names are omitted, mGetSelection returns the contents of all the fields of gTable, in their order of creation. Fields of type Media are ignored by mGetSelection.

Besides gTable, all other parameters are optional. However, if a parameter is present, all its preceding ones must also be present. For example, if #recs is present, the result type and From parameters must also be present.

mGetField requires that you set the current record to the record you need to retrieve data from. mGetSelection does not.

See Appendix 2: mGetSelection Examples for complete examples of each of the above variations of mGetSelection.

Note Because it uses the selection's Master Field, it is recommended that you call mOrderBy with the appropriate field before calling mSelect and mGetUnique.
If you don't call mOrderBy, mGetUnique returns unique values from the Master Field chosen by default by INM V12 Database, which is either the one indexed by the default index (if the table was just opened), or the one indexed by the best index chosen by INM V12 Database for the last selection. (See Selection and current record.)

Read unique values of a field

mGetUnique returns unique values of the Master Field in a string or a Lingo list (See Selection and current record above for a definition of Master Field).

Syntax:

```
a = mGetUnique(gTable, "literal")
b = mGetUnique(gTable, "list")
```

mGetUnique is very convenient to populate a user interface element (such as scrolling list or pull-down menu) with search values that are relevant only for a specific database and context.

Example: In a clothing catalog, you want to display only the available colors for a specific category and size of product (e.g., T-shirt and XXL). You run this script:

```
mSetCriteria(gTable, "category", "=", "T-shirt")
mSetCriteria(gTable, "and", "size", "=", "XXL")
mOrderBy(gTable, "color")
mSelect(gTable)
put mGetUnique(gTable, "literal") into field "ScrollList"
```

This script retrieves unique values of the "color" field (which is the Master Field) to the field "ScrollList". Assuming that your selection contains 30 records (10 with Color = "Red", 10 with Color = "Green" and 10 with Color = "Blue"), the above script puts the string:

```
Blue
Green
Red
```

in field "ScrollList".

Running this script:

```
mSetCriteria(gTable, "category", "=", "T-shirt")
mSetCriteria(gTable, "and", "size", "=", "XXL")
mOrderBy(gTable, "color")
mSelect(gTable)
put mGetUnique(gTable, "list") into field "ScrollList"
```

returns the list:

```
[ "Blue", "Green", "Red" ]
```

Data formatting

mDataFormat assigns a display pattern to a field so that all data read from that field are formatted according to that pattern. All INM V12 Database methods that read data from a formatted field are affected. These include mGetField and mGetSelection

Syntax:

```
mDataFormat(gTable, FieldName, Pattern)
```

This example forces all data retrieved from the field price to be formatted with 3 integral digits and 2 decimal places.

Example:

```
mDataFormat(gTable, "price", "999.99")
```

mDataFormat can be applied to fields of type float, integer and date. Media and string fields cannot be formatted.

To reset the formatting of a pattern to its original value, call mDataFormat with an empty string.

Example:

```
mDataFormat(gTable, "price", "")
```

Format integers and floats

Valid patterns for fields of type integer and float contain:

- 9 designates a digit at that position (possibly 0),
- # designates a digit or a space at that position,
- . (period) designates the decimal point,
- any other character is interpreted literally.

This example forces the output of the field ratio to 2 integral digits, 2 decimal places and a trailing "%" sign:

```
mDataFormat(gTable, "ratio", "99.99%")
put mGetField(gTable, "ratio")
```

If the value in field ratio is 34.567, the displayed string is "34.57%".

The pattern "###9999" forces the output of an integer field to be formatted with no less than four digits and with three leading spaces, if necessary. Thus:

```
4           is formatted as    "   0004"
123         is formatted as    "   0123"
314159      is formatted as    " 314159"
3141592     is formatted as    "3141592"
31415926    is formatted as    "#######"
```

The last formatting in the above example fails because an eight-digit integer does not fit in a seven-digit pattern.

The pattern "(999) 999-9999" is convenient for formatting phone numbers stored as integers. For example:

```
mDataFormat(gTable, "phone", "(999) 999-9999")
put mGetField(gTable, "phone")
■  returns something formatted as "(514) 871-1333"
```

Format dates

Valid patterns for fields of type date are combinations of:

- D for days,
- M for months,
- Y for years,
- any other character is interpreted literally.

This example formats the date in the "Year-Month-Day" numerical format:

```
mDataFormat(gTable, "TheDate", "YY-MM-DD")
put mGetField(gTable, "TheDate")
```

Assume the content of field TheDate for the current record is April 30, 1945 – the returned string is "45-04-30".

Ds, Ms and Ys can be combined in the following way:

To format	Use this

	sequence
Days as 1-31	D
Days as 01-31	DD
Weekdays as Sun-Sat	DDD
Weekdays as Sunday-Saturday	DDDD
Months as 1-12	M
Months as 01-12	MM
Months as Jan-Dec	MMM
Months as January-December	MMMM
Years as 00-99	Y or YY
Years as 1900-9999	YYY or YYYY

Examples:

The pattern	Formats 5 January 1995 as
D	5
DDDD	Thursday
MM	01
DD-MM	05-01
MMM DD, YY	Jan 05, 95
On D MMMM, YYYY	On 5 January, 1995
'Weekday='DDDD; 'Month=' MMMM	Weekday=Thursday; Month=January

In this last example, apostrophes around 'Weekday' and 'Month' are mandatory, otherwise the "d" in *Weekday* and the "m" in *Month* would interfere with the pattern itself. To specify real apostrophes within date patterns, use two consecutive apostrophes.

When a table is first opened, the default format of all its Date fields is set to "YYYY/MM/DD".

By default, the names for the months in INM V12 Database are (MMMM)

```
January, February, March, April, May, June, July, August, September,
  October, November, December
```

The short names for the months are (MMM)

```
Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
```

The names for the weekdays are (DDDD)

```
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
```

The short names of the weekdays are (DDD)

```
Sun, Mon, Tue, Wed, Thu, Fri, Sat
```

All of these names can be replaced by custom names through the properties of the database (see Properties of Databases).

> **Note:** If a formatting pattern is assigned to a field, all values retrieved from that field become strings (see Typecasting in Database Basics).

Read fields of type media

`mGetMedia` retrieves data from fields of type `media` and stores them directly in the designated Director member.

Syntax:

```
mGetMedia(gTable, fieldName, DirMember)
```

Example:

```
mGetMedia(gTable, "photo", member "PhotoMember")
```

This example stores the content of the field "photo" from the current record into the member named "PhotoMember" in Director's internal castlib. If more than one castlib is used in a project, `mGetMedia` can also retrieve media to any castlib through this syntax:

Example:

```
mGetMedia(gTable, "photo", member 28 of castlib "album")
```

Add records to a database

To add a new record to a INM V12 Database table, use `mAddRecord`, then call `mSetField` as many times as needed (typically once for each field of your table) and finally call `mUpdateRecord`.

In this example, a new record is created for the item "goggles" and its price is set to $158.99:

```
mAddRecord(gTable)
mSetField(gTable, "ItemName", "Goggles")
mSetField(gTable, "Price", 158.99)
mUpdateRecord(gTable)
```

If `mUpdateRecord` is not called, the record created with `mAddRecord` is not saved to the database. After calling `mUpdateRecord`, the record is created and kept in a cache: it is not immediately written to disk. Thus, if the computer crashes or a power failure occurs, the database file on disk may become corrupt. To ensure that the newly added records are saved onto the hard disk, Flush the database to the disk (see FlushToDisk).

New records are always added to the end of the selection regardless of the criteria used to form the selection.

> **Note:** Only `mSetField` and `mSetMedia` can be called after `mAddRecord` and before `mUpdateRecord`. Calling any other method aborts the new record adding process and sets the current record to the previous current record.
> Thus, if you started to add a record and wish to abort the operation, simply call `mGetField` instead of calling `mUpdateRecord`.

Update data in a database

Writing data is very similar to reading data. Writing data is accomplished with `mSetField`. Prior to updating a field, you must set the current record, and your intentions must be indicated in INM V12 Database with `mEditRecord`. Once this is completed, INM V12 Database will update your database with `mUpdateRecord`.

After calling `mUpdateRecord`, the modified record is kept in a cache: it is not immediately written to disk. Thus, if the computer crashes or a power failure occurs, the database file on disk may become corrupt. To ensure that the newly added records are saved onto the hard disk, Flush the database to the disk (see FlushToDisk).

Note: Updating the contents of fields that have a full-index may take more time to write to the database than equivalent fields without full-indexes.

Write to fields of type Integer, Float and String

In this example, the name of the current record is changed to "funnel" and its price to $2.95:

```
mEditRecord(gTable)
mSetField(gTable, "name", "funnel")
mSetField(gTable, "price", 2.95)
mSetField(gTable, "CatalogNumber", 1234)
mUpdateRecord(gTable)
```

As with `mAddRecord`, every call to `mEditRecord` must be balanced with a call to `mUpdateRecord` and the only methods that should be used in between are `mSetField` and `mSetMedia`. Calls to `mSetField` will result in an error if not preceded by `mEditRecord`. Calling `mGetField` after `mEditRecord` and before `mUpdateRecord` will abort any changes made to the record.

If an error occurs when updating a record (e.g. Duplicate Key error in a given field), none of the preceding calls to `mSetField` are taken into consideration.

When writing to a field whose type is not the same as the supplied parameter, INM V12 Database tries to cast the parameter to the appropriate type and to interpret it as accurately as possible (see Typecasting).

Example:

```
mSetField(gTable, "price", "3.14") -- stores the value 3.14
mSetField(gTable, "price", "xyz") -- stores the value 0.00
mSetField(gTable, "name", 1234) -- stores the string "1234"
```

Write to fields of type Date

Writing to a field of type Date is similar to writing to field of type Integer, Float or String, except that INM V12 Database requires the date to be supplied in raw format (YYYY/MM/DD).

Example

```
mSetField(gTable, "BirthDate", "1993/02/22") -- is valid
mSetField(gTable, "BirthDate", "02/22/1993") -- is not valid
```

Storing the current date in Raw format may be difficult as Lingo's `the Date` function returns the current date as formatted in the Control Panel settings of the computer it is running on. In this case, use `mGetProperty(gDB, "CurrentDate")` to retrieve the current date in raw format (see CurrentDate in Properties of Databases).

Write to fields of type Media

To write any media type, call:

```
mSetMedia(gTable, MediaField, DirMember)
```

This example copies the content of member "Yeti" of castlib "photo_album" to the field `photo`:

```
mEditRecord(gTable)
mSetMedia(gTable, "photo", member "Yeti" of Castlib "photo_album")
mUpdateRecord(gTable)
```

Delete a record

Call `mDeleteRecord` to delete the current record.

Syntax:

```
mDeleteRecord(gTable_instance)
```

Example:

```
mDeleteRecord(gTable)
```

After calling `mDeleteRecord`, the record that follows the record being deleted becomes the new current record. If no record follows the deleted record, the preceding record becomes the new current record. If no record precedes the deleted record, the selection is then empty and the current record is not defined.

Note: Use this method with caution. There is no way to *undelete* records in INM V12 Database. As a general rule, avoid giving direct access of this method to the end-user through your user interface.

Delete all the records of a selection

Call `mSelDelete` to delete all the records of a selection at once.

Syntax:

```
mSelDelete(gTable)
```

After `mSelDelete` has been completed, the selection is empty and the current record is undefined.

Search data with mSetCriteria

When searching data, you often need to isolate a specific group of records that satisfy a common condition in a table. These conditions are called **search criteria** and the subset of isolated records is the **selection** (see Database Basics for an explanation of selections and current records). `mSetCriteria` is the method used to specify search criteria, followed by `mSelect` to trigger the search process.

Syntax:

```
mSetCriteria(gTable, FieldName, operator, Value)
mSelect(gTable)
```

Simple search criteria

A search criterion has at least three characteristics:

- `FieldName`: this is a valid field name in the table instance,
- `operator`: this is a comparison keyword. Valid operators are `=`, `<`, `<=`, `>`, `>=`, `<>`, `starts`, `contains`, `wordStarts` and `wordEquals`.
- `value`: this is the value to which the field contents must be compared, in order to be selected.

This example selects all items that are cheaper that $12.

```
mSetCriteria(gTable, "price", "<", 12)
mSelect(gTable)
```

Upon completion of `mSelect`, the resulting selection contains the set of records that satisfy the defined criteria. In the above example, all records that contain a `price` field with a value that is strictly smaller than *12* are selected. In addition, the selection is sorted with an increasing order of prices given that a search with a defined ascending index was performed on that field.

The current record is the first record of that selection. In our example, it is the least expensive item.

If you want the selection sorted in an order other than the one proposed by `mSelect`, you can do so by calling `mOrderBy` right before calling `mSelect`. However, keep in mind that this may cost some additional processing time.

Values provided to `mSetCriteria` need to be in the same type as `FieldName`. As discussed in Database Basics / Typecasting, INM V12 Database tries to automatically typecast `value` to the proper type. Borderline conditions such as criteria containing extra spaces, carriage returns or other unwanted characters must be avoided.

Example:

```
mSetCriteria(gTable, "price", "<", "100")
```

is strictly equivalent to

```
mSetCriteria(gTable, "price", "<", 100.00)
```

but beware of the unpredictable results of

```
mSetCriteria(gTable, "price", "<", "..100.00..")
```

Operations on fields of type `Date` require that Value be supplied in raw format (see Step 2: Prepare the Data / Dealing with dates). This example locates all records where field `theDate` contains a date occurring before May 21st, 1997.

```
mSetCriteria(gTable, "theDate", "<", "1997/05/21")
```

Sort a selection (mOrderBy)

You can define a sort order on a selection by calling the `mOrderBy` method prior to calling `mSelect`. Specify the sorting order (whether `ascending` or `descending`) and the field upon which the sort is performed.

Example:

```
mSetCriteria(gTable, "ItemName", "contains", "hat")
mOrderBy(gTable, "price", "descending")
mSelect(gTable)
```

The above example selects all hats in `gTable` and returns a selection sorted by a descending order of prices (most expensive to least expensive).

If `mOrderBy` is not called before calling `mSelect`, the sort order of the selection depends on the index used to perform the search. That index is automatically chosen by INM V12 Database to optimize the search time. See the **V12 Methods Reference manual: mOrderBy**.

Operators

Following is a list of valid operators and their meanings. Although comparisons of `integer`s, `float`s and `date`s are straightforward, comparing strings and custom string types depends on how those comparison rules are defined. For details on various character sets and custom string types, please see the appendices in this manual.

Media fields cannot be compared.

Equal (=)

The "=" operator is used to locate data that exactly match the specified value.

Example:

```
mSetCriteria(gTable, "price", "=", 3.14)
```

specifies a search for items that cost exactly $3.14 .

Example:

```
mSetCriteria(gTable, "ItemName", "=", "hat")
```

specifies a search for items named "hat". Items named "hats" or "hatchet" will not be selected. Since INM V12 Database does not differentiate upper case and lower case characters, items named "HAT" or "Hat" will be selected.

Not Equal (<>)

The "<>" operator has the opposite effect of the "=" operator. It is used to locate data that are different than the specified value.

Example:

```
mSetCriteria(gTable, "price", "<>", 9.99)
```

specifies a search for all items except those that cost $9.99.

Less than (<)

The "<" operator is used to locate data that are strictly smaller that the specified value.

Example:

```
mSetCriteria(gTable, "price", "<", 10)
```

specifies a search for items that cost less than $10. Items that cost exactly $10 are not selected.

Example:

```
mSetCriteria(gTable, "ItemName", "<", "hat")
```

specifies an alphabetical search for items with names that precede the letter “h” in "hat". This includes "cap", "bonnet" but excludes "hat".

Less or equal (<=)

The "<=" operator is used to locate data that are smaller or equal to the specified value.

Example:

```
mSetCriteria(gTable, "price", "<=", 10)
```

specifies a search for items that cost no more than $10.

Example:

```
mSetCriteria(gTable, "ItemName", "<=", "hat")
```

specifies an alphabetical search for items with names that precede equal “h” in "hat". This includes "cap", "bonnet" and "hat".

Greater than (>)

The ">" operator is used to locate data that are strictly larger than the specified value.

Example:

```
mSetCriteria(gTable, "CatalogNumber", ">", 1000)
```

This example specifies a search for items with catalog numbers larger than 1000. Catalog number 1000 will *not* be selected.

Example:

```
mSetCriteria(gTable, "birth date", ">", "1961/12/31")
```

This example specifies a search for records with a "birth date" field occurring *after* Dec 31st, 1961, (excluding that date). The earliest birth date in the selection should be Jan 1st, 1962 or later.

Greater or equal (>=)

The ">=" operator is used to locate data that are larger or equal to the specified value.

Example:

```
mSetCriteria(gTable, "CatalogNumber", ">=", 1000)
```

This example specifies a search for items with catalog numbers larger or equal to 1000. Catalog number 1000 *will* be selected.

Example:

```
mSetCriteria(gTable, "birth date", ">=", "1961/12/31")
```

This example specifies a search for records with a "birth date" field occurring *on* or *after* Dec 31st, 1961. Therefore, the earliest birth date in the selection may be Dec 31st, 1961.

Starts

The "starts" operator can be used with fields of type `string` only (including custom string types). It locates records that start with a given sub-string in the specified field.

Example:

```
mSetCriteria(gTable, "description", "starts", "hat")
```

This example identifies items for selection with descriptions such as "**Hat** with two propellers" and "**Hat**chet for heavy-duty applications".

If an index is defined on the field `description`, the search process is very fast. If not, the search takes more time but can be performed nonetheless.

Contains

The "contains" operator can be used with fields of type `string` only (including custom string types). It locates records that contain a given sub-string in the specified field.

Example:

```
mSetCriteria(gTable, "description", "contains", "hammer")
```

sets records for selection containing descriptions such as "the greatest hammer in the world" and "casing for hammers of all sizes".

Searches using the "contains" operator are inherently sequential. They cannot take advantage of any index definition and can be very slow.

WordStarts

The "wordStarts" operator can be used only with fields of type `string` (including custom string types) with defined full-indexes. It locates records that contain words that fully or partially match the value specified to `mSetCriteria`.

Example:

```
mSetCriteria(gTable, "description", "wordStarts", "ham")
```

This example would identify records containing descriptions such as "Gigantic **ham**burger with fries" and "The greatest **ham**mer in the world". It would not find records containing descriptions such as "Champion" or "Gotham City" because the words in these records don't *start* with the sub-string "ham".

Since "wordStarts" operates on full-indexes, searching is performed very quickly.

Note: Although words such as "hamburger" and "hammer" can be quickly found by the example query, the word "ham" will never be found because it is shorter than the minimum word length set for full-indexing, and therefore is not stored in the index.

WordEquals

The "wordEquals" operator can be used only with fields of type `string` with defined full-indexes. It locates records that contain words that fully match the value specified to `mSetCriteria`.

Example:

```
mSetCriteria(gTable, "description", "wordEquals", "form")
```

This example identifies records containing descriptions such as "claim **form**". Records containing words such as "forms" or "formalism" would not be selected.

Since "wordEquals" operates on full-indexes, searching is performed very quickly.

Difference between ‘Contains’ and ‘WordStarts'

Why should you bother using the slow "contains" operator if "wordStarts" does the job faster?

Because "wordStarts" requires that a full-index be defined on a field. Full-indexes allow for quick searches, but require more disk space and more time when updating data.

Another reason is that "wordStarts" can only search for *words* that match or begin with a given string. For example, if the description field of a certain record contains the text "Dark chocolate with hazelnuts":

```
mSetCriteria(gTable, "description", "contains", "cola")
```

would locate that record ("chocolate" contains the sub-string "cola"), whereas

```
mSetCriteria(gTable, "description", "wordStarts", "cola")
```

would not. This is because no word in the description field *starts* with the string "cola".

Note: Words shorter than the minimum word length set for full-indexing cannot be looked for with the "wordEquals" or "wordStarts" operators. In such cases, you must use the "contains" operator instead.

Complex search criteria

`mSetCriteria` can also be called with four parameters. The additional parameter is the Boolean operator "AND" or "OR". It is added to the second call to `mSetCriteria` and inserted before the field to be searched.

Example:

```
mSetCriteria(gTable, "name", "contains", "hat")
mSetCriteria(gTable, "and", "price", "<=", 50)
mSelect(gTable)
```

The above example selects all hats up to $50.00 in the table referred to by `gTable`.

The first call to `mSetCriteria` should use three parameters, and it can be chained with as many four-parameter calls as needed to specify your query. Using `mSetCriteria` with three parameters will reset and ignore the preceding search criteria.

Another example, using the Boolean "OR" operator is:

```
mSetCriteria(gTable, "name", "contains", "hat")
mSetCriteria(gTable, "or", "name", "contains", "helmet")
mSetCriteria(gTable, "or", "name", "contains", "cap")
mSelect(gTable)
```

It selects all records whose "name" fields contain either "hat" or "helmet" or "cap".

Complex criteria are very powerful but can be tricky to use. This example illustrates complex criteria.

```
mSetCriteria(gTable, "name", "=", "hat")
mSetCriteria(gTable, "or", "name", "=", "helmet")
mSetCriteria(gTable, "and", "price", "<=", 50)
mSelect(gTable)
```

This section of script selects all hats priced under $50.00 and all helmets under $50.00. This is very different from:

```
mSetCriteria(gTable, "name", "=", "hat")
mSetCriteria(gTable, "and", "price", "<=", 50)
mSetCriteria(gTable, "or", "name", "=", "helmet")
mSelect(gTable)
```

where the selection consists of all hats under $50.00 and all helmets listed at any price.

To illustrate the semantic difference between the two requests, we could express the first as:

```
(name = "hat" or name = "helmet") and price <= 50
```

whereas the second could be written as:

```
(name = "hat" and price <= 50) or name = "helmet"
```

Note: Complex searches that use the OR operator are always slower than those that use the AND operator. This is true with INM V12 Database as well as with most other database management systems.

Important: The current version of INM V12 Database does not have the ability to perform searches such as

```
name = "hat" or (name="helmet" and price<=50)
```

(note the parentheses).

The first two criteria are always grouped first and the third criterion is added to the result.

Partial Selections

The selection process can be time-consuming if a large number of records match the criteria you specify. The worst-case scenario is when all the records of a table match the specified criteria. This can handicap your project if you have no control over the queries the end-user can express.

To speed up the selection process, you can limit the number of records INM V12 Database places in the selection with this syntax of `mSelect`.

```
mSelect(gTable, from, #recs)
```

Example:

```
mSetCriteria(gT, "LastName", "=", "Smith")
mSelect(gT, 1, 100)
```

The above example returns up to a maximum of 100 records in the selection, regardless of the total number of Smith's in the database. If less than 100 Smith's exist, all of them are selected.

To retrieve the next 100 records that contain "Smith" in the "LastName" field, call:

```
mSelect(gT, 101, 100)
```

Note Partial selections also work with complex searches, but not *all* of them. They are only accepted for complex searches that *do not* use full-text indexes (i.e., `WordStarts` or `WordEquals`).

Check the size of a selection

It is sometimes useful to know the number of items localized in a selection. This is the purpose of the `mSelectCount` method.

Example:

```
mSetCriteria(gTable, "name", "=", "hat")
mSelect(gTable)
selSize = mSelectCount(gTable)
```

In this example, the number of records in the selection (the number of items named "hat") is stored in `selSize`.

Styled text management

Cast members of type **Text** keep all the formatting styles assigned to their content (including fonts, colors, margins, etc.) and can be anti-aliased. They can be styled and edited both at authoring time and runtime.

INM V12 Database manages Director Text members through fields of type String, Integer, Float and Date. Both types of Text members are split in two parts: the text of the member is stored in the V12 field itself. The media component — whether bitmap or binary representation of styled text — is stored in a hidden field of type `media`. The text component is used to searching and sorting, whereas the media component is used for storage and retrieval.

Any INM V12 Database field of type `integer`, `float`, `date` or `string` can be used to store a text member through this syntax:

```
mSetField(gTable, fieldName, member)
```

For example, assume that `gTable` contains a field of type `string` named "Banner" and that member 28 is a Text member containing the anti-aliased text "The Tiger in your Engine",

```
mSetField(gTable, "banner", member 28)
```

stores the styled Text member 28 in the field "Banner". Technically, the string "The Tiger in your Engine" is stored in the field "Banner" and the media component of the Text member is stored in a hidden Media field.

To retrieve styled text, call `mGetMedia` as follows:

```
mGetMedia(gTable, "Banner", member "myBanner")
```

This instruction retrieves the banner image from the V12 Database file and places it in the member named "myBanner". Note that the V12 Database field "Banner" mentioned above is of type `string` and not `media`.

Alternatively, you can retrieve the content (the individual characters) of the data contained in the field "Banner" as follows:

```
aText = mGetfield(gTable, "Banner")
-- assigns "The Tiger in your Engine" to aText
```

If, for some reason, your script does the following:

```
mSetField(gTable, "banner", member 28)
mSetField(gTable, "banner", the text of member 28)
```

the second call to `mSetField` would replace the content of the field "Banner" with the unformatted text contained in member 28. This would replace any image associated with that specific record/field by an empty image. A subsequent call to `mGetMedia` would then return the placeholder.

Searching and Sorting Styled Text Fields

As a result of this technique, it is possible to search and sort fields that contain styled text based on the content component of the fields. Since the fields used to store styled text are of type `string`, `integer`, `float` or `date`, operations such as indexing, sorting, searching, etc. all remain valid.

Exporting Data

`mExportSelection` allows exporting of data from a INM V12 Database table to TEXT or DBF files (DBase III). Only the selected records are exported (i.e. those in the selection). To export a complete table, make sure it is entirely selected first (see Selection and current record and Select all the records of a table).

Exporting in TEXT Format

The syntax for exporting all the fields of a table's selection is:

```
mExportSelection(table_instance, "TEXT", FileName)
```

The above instruction exports all the fields of the selection to the file named `FileName`. The field and record delimiters are `TAB` and `CARRIAGE_RETURN` respectively.

To specify custom field and record delimiters, use:

```
mExportSelection(table_instance, "TEXT", FileName, FldDelimiter,
   RecDelimiter)
```

Example:

```
mExportSelection(gTable, "TEXT", the moviePath&"Output.txt", "~",
   "%")
```

This example exports the selection in a text file named "Output.txt" with the field delimiter "~" and the record delimiter "%".

`mExportSelection` can also export only selected fields in the following way:

```
mExportSelection(table_instance, the moviePath&"TEXT", FileName,
   FldDelimiter, RecDelimiter, Field1, Field2, ...)
```

Example:

```
mExportSelection(gTable, "TEXT", "Data.TXT", TAB, RETURN, "ItemName",
   "catalog number", "price")
```

This example exports the selection in a text file named "Data.TXT" with TAB and RETURN delimiters. The only exported fields are `ItemName`, `catalog number` and `price`, in that order.

The first line in the exported file contains the names of the exported fields separated by the selected field delimiter. The resulting text file is in the character set of the current Operating System (this is relevant only if accented characters are present in the exported data).

`mExportSelection` takes the format patterns specified in `mDataFormat` into account. The sorting order of the exported records is identical to the one set on the selection. Media fields are ignored during the exporting process.

Exporting in DBF Format

The parameters for exporting DBF files are identical to those of exporting text, without the field and record delimiters.

Example:

```
mExportSelection(gTable, "DBF", "Goliath.DBF")
-- exports all fields of gTable
```

Or:

```
mExportSelection(gTable, "DBF", "Goliath.DBF", "ItemName", "catalog
   number", "price")
-- exports only fields ItemName, catalog number and price.
```

These rules apply when exporting to a DBF file format

- `String` fields are exported to fields of type Character, if the buffer size of the string field is declared to be no larger than 255 characters. Otherwise, they are exported to field of type Memo.
- `Integer` fields are exported to fields of type Numeric.
- `Float` fields are exported to fields of type Numeric with 10 digits after the fixed point.

- `Date` fields are exported to fields of type Date.
- `Media` fields are ignored.

Note: Although INM V12 Database can read all kinds of DBF file variants, it exports data only in the popular DBase III format. This is mainly because Dbase III is universally read by all DBF-compliant systems whereas other more recent formats are not.

Cloning a Database

Cloning a database makes a copy of an existing database file, with all the table, field and index definitions but with none of the data. This is similar to creating a database file from a template rather than starting a new project. Contrary to creating a database with `mReadDBstructure` (which requires an INM V12 Database license to create legal INM V12 Database files) this method can be used at runtime.

Syntax:

```
mCloneDatabase(db_instance, new_pathname)
```

Example

```
mCloneDatabase(gDB, the MoviePath & "myClone.V12")
if CheckV12Error() then exit -- don't continue if clone failed
gDB_cloned = New(Xtra "V12dbe", the MoviePath & "myClone.V12",
   "ReadWrite", "pwd")
```

In this example, a new database file named "myClone.V12" is created using the same tables, fields and index definitions, as well as the same password as the database file designated by the global variable `gDB`. This implies that the original database file, designated by the variable `gDB`, must be opened with the appropriate password before cloning. After the new database has been cloned, you need to create an instance of it, `gDB_cloned`, using the `New` method.

Freeing up Disk Space (packing)

Most database management systems, including INM V12 Database, do not reclaim the space freed by deleted records, for the sake of performance. Consequently, as records are created and deleted, the size of the database grows continuously. `mPackDatabase` can be used periodically to reclaim lost bytes.

Syntax:

```
mPackDatabase(database_instance, NewFilePathName)
```

Example:

```
mPackDatabase(gDB, the MoviePath & "Packed_DB.V12")
```

This example compresses `gDB` into a new file named `Packed_DB.V12` located in the same folder as the current Director movie.

At the end of the operation, `database_instance` stays valid (referring to the non-packed database) and `NewFilePathName` is a new file that can be opened with INM V12 Database.

If you just need to compress your current database without creating a new file, you can do so by compressing it into a new temporary database, deleting your initial database and renaming the temporary database to your initial database's name. `FileXtra`, a free

Xtra delivered with recent versions of Director (also available for download at http://www.kblab.net/xtras/), comes in handy as shown below:

```
-- let fName be your V12 database's name
-- first make sure to close all table instances
gT.mClose()
gT = 0
mPackDatabase(gDB, the MoviePath&"temp.V12")
if CheckV12Error() then exit -- don't continue if pack failed
DeleteFile(the MoviePath&fName)
RenameFile(the MoviePath&"temp.V12", the MoviePath&fName)
```

Fixing Corrupted Database Files

Databases may become corrupt if a power failure or system crash occurs while updating records. Therefore, INM V12 Database is unable to reopen the database and returns an explicit error code when trying to create a database instance.

Some of these corrupt databases can be fixed with mFixDatabase. The syntax for mFixDatabase is:

```
mFixDatabase(Xtra "V12dbe", pathname, new_pathname)
```

pathname is the name of the database to fix and new_pathname is the name of the fixed database, which may reside on a different volume.

mFixDatabase is a static method (its first parameter is the Xtra library itself, not on an instance of V12dbe). In this example:

```
mFixDatabase(Xtra "V12dbe", "Crash.V12", "Recovered.V12")
```

mFixDatabase tries to read data from "Crash.V12" and saves the data to "Recovered.V12".

Note: mFixDatabase attempts to save a corrupted file as much as possible, but there is no guarantee on the result. mFixDatabase essentially attempts to rebuild the indexes of a damaged INM V12 Database file, but if the file's headers or data clusters are damaged, chances are that the recovery process will fail.

Checking the Version of the Xtra

At authoring time, you check the INM V12 Database Xtra's version by opening its Get Info window in the MacOS Finder, or by checking its Properties in Windows' Explorer.

Both at authoring time and runtime, you can call mXtraVersion to retrieve the version of the Xtra. Example:

```
v = mXtraVersion(xtra "v12dbe")
put v -- puts "V12,3.3,Multi-User" in message window
if (char 1 of item 2 of v) <> 3 then Alert "not version 3"
```

Changing a Password

You can change the password assigned to a database by using the mSetPassword method. The new password can be an empty string. The syntax is as follows:

```
mSetPassword(gDB, oldPassword, newPassword)
```

Example:

```
mSetPassword(gDB, "houdini", "ali baba")
```

Dynamically Downloading Databases via the Internet

V12 has the ability to dynamically download an updated database via the Internet to a local storage device. `V12Download` is the method that you use to replace a database on a user's local drive.

The syntax is as follows:

```
V12Download(url, local_file, password, completion_handler,
  status_handler, ref)
```

`V12download` resumes and returns control to Director immediately after initiating the download query. It calls `status_handler` as frequently as possible during download (generally used to display the download status in the user interface) and ends-up calling `completion_handler` when the download is complete (generally used to open the database and start working with it).

If a local V12 database of the same name already exists, the downloaded file replaces it. The Xtra automatically ensures that it is a valid V12 database and its password is supplied and correct.

For an example of its use, see the **V12 Methods Reference: V12Download.**

`V12DownloadInfo` is a method that you can use to determine the size of the database to be downloaded. This can be useful in determining whether or not the database has changed since the last time it was downloaded. Due to the fact that not all HTTP servers support time and date stamps on files, this method helps you diagnose the status of the database files.

Tip: You may also want to consider having an associated text file or a separate database with a time stamp or other important information within it, which you can use to determine whether or not the database has changed or needs to be downloaded.

Both `V12download` and `V12DownloadInfo` require that the Director Networking Xtras be installed (namely InetURL, Netfile and NetLingo). To do so, in Director, choose MODIFY->MOVIE->XTRAS... Add Network.

Tip: V12 Online Companion is an optional add-on set of Xtras for V12 that allows you to dynamically query databases on remote servers via the Internet.

The major difference between the `V12Download` functionality and V12 Online is that V12 Online allows you to execute queries on a database residing on a remote host. `V12Download` must download the complete database and store it for use on the user's local hard drive.

Tip: If you need to download a V12 Database from a URL that is behind a proxy server or via Secure HTTP, use INM SecureNet Xtra, which contains a method named snxDownloadNetThing and supports proxy servers and Secure HTTP from Projectors. The snxDownloadNetThing method is not supported in Shockwave movies:

http://www.INM.com/products/snxdirector/ .

Errors and Defensive Programming

Error Management in Applications

Effective error management is a key to any reliable script or program. If you choose to implement your project with the INM V12 Database Behaviors Library, you automatically take advantage of this Library's efficient built-in error management. You just need to make sure that the "Show Alert on Error" check box is checked when dragging/dropping a Behavior.

INM V12 Database's Lingo interface provides methods that allow you to keep a close check on your programming. Use the global functions `V12Status()` and `V12Error()`, to confirm each step of database creation and handling.

As well, Director has two interesting tools to help you detect your Lingo scripting errors: the **Watcher** and the **Debugger**, both available in the Windows menu. INM V12 Database's error detecting functions, the Watcher and the Debugger can be used together to efficiently debug your scripts. Look for "debugging" in Director's Help files or User Manual's index for details.

Also, a good tool to take advantage of, in database projects that write data to a storage device, is FlushToDisk. It will help you ensure that all of your data is written to the storage device at important intervals, flushing the disk cache.

Checking the Status of the Last Method Called

Call `V12Status()` after each call to a INM V12 Database method (both V12dbe and V12table methods) to check its outcome. `V12Status()` returns *0* if no error occurred during the execution of that method. Otherwise, it returns a non-zero error code.

Example:

```
aPrice = mGetField(gTable, "price")
errCode = V12Status()
if errCode <> 0 then
  Alert("Mayday! Mayday! mGetField returned error code" && errCode)
end if
```

If `V12Status()` returns a non-zero result, you can call `V12Error()` to get the details of the error. When called with no parameter – as in `V12Error()-` this global function returns a plain-English explanation of the outcome of the last called method. If an error occurred in that last call to INM V12 Database, `V12Error()` provides a detailed contextual report on it.

Example:

```
aPrice = mGetField(gTable, "price")
errCode = V12Status()
if errCode <> 0 then
  Alert ( V12Error() )
end if
```

CheckV12Error

The `CheckV12Error()` Lingo handler is often used throughout this manual and in sample projects. It is a generic error handling routine that centralizes all the error management logic in a single handler. That way, it can more easily be adapted from one project to the other, or from a *debugging* mode to a *delivery* mode (e.g. the debugging mode would display alerts, whereas the delivery mode would, in addition, write an error log to an external file with the FileIO Xtra).

```
on CheckV12Error
  if V12Status() then
     alert V12Error()
     return TRUE
  end if
  return FALSE
end CheckV12Error
```

Note: Usually, you call `V12Status()` to get an error or warning code, then `V12Error()` to get a full explanation of that error or warning. Alternatively, your application can choose to handle specific error codes uniquely.

Example:

```
mSetCriteria(gT, "City", "=", "Paris")
CheckV12Error()
mSelect(gT)
CheckV12Error()
```

Optionally, you can take advantage of `CheckV12Error`'s return value to decide how to continue the execution of your handler.

Example:

```
gDB = new (Xtra "V12dbe", the moviePath&"myDatabase.V12",
  "ReadWrite", "secret password")
if NOT CheckV12Error() then  -- only create the table if the DB
  exists
  gT = new (Xtra "V12table", mGetRef(gDB), "myTable")
end if
```

This script attempts to create a table instance only if the database instance was successfully created.

Errors and Warnings

Typically, two types of faults can occur in using INM V12 Database:

- **Errors**, which lead to major problems that require that you, stop the execution of your script.
- **Warnings**, which happen while executing certain instructions partially or in borderline conditions that you need to be aware of.

An example of an error is *File not found*, when trying to import data. When a file is not found, it does not make sense to continue the importing operation until the problem is solved.

An example of a warning is *No previous record*, when trying to go to the previous record from the first record of the selection. In such a case, the current record remains valid (although unchanged).

V12Status() returns negative codes for errors and positive codes for warnings. Often, the term *error* is used to designate faults of both types (i.e. errors and warnings).

Based on the value returned from V12Status() you can display specific messages to your end-users.

Example:

```
mSetCriteria(gTable,fName, "contains", fMyWord)
errCode = V12Status()
if errCode = -8690 then    -- empty field name
               Alert(“You must choose a field to search”)
Else if errCode <> 0 then
  Alert("msetCriteria returned error code: " && errCode)
Else mSelect(gTable)
End if
```

In this example the application advises the user to select a field before doing the search operation.

For a complete listing of all V12 error codes, consult the sections named V12 Database Methods and V12 Database Error Codes.

Using the Verbose Property

The verbose property offers a convenient way to monitor and debug your application’s interactions with a V12 database during the testing phase. When verbose is turned “on”, both successful v12 method calls and errors are reported to Director’s message window. Turn verbose “on” at the beginning of a script segment that includes several INM V12 Database method calls; then turn it “off” at the end of that section of code. Although verbose allows you to monitor database activity while in authoring and testing mode, it should not substitute for checking V12Status and handling runtime errors.

Make sure all calls to mSetProperty(gDB, "verbose", "on") are deleted or commented out before distributing your application. For more information on the verbose property see Verbose in Properties of Databases.

Preventing Data Corruption Due to Script Errors

When a scripting error occurs (e.g. Lingo syntax error), Macromedia Director displays an error message and aborts the execution of the current handler. This can be annoying in cases where local database and table instance variables are defined in your handler.

Example:

```
on doSomethingCritical
  gDB = New(Xtra "V12dbe", the MoviePath&"catalog.V12", "ReadWrite",
  "top top top")
  gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")
  -- other Lingo statements
  -- with a syntax error somewhere which
  -- causes the handler to abort
   gTable.mClose()
  gTable = 0
   gDB.mClose()
   gDB = 0
end doSomethingCritical
```

In this example, a Lingo error is detected and the database remains open. If the database was opened in `ReadWrite` mode and changes were made, the database file might become corrupted.

You can prevent this problem by declaring the instance variables as global variables.

Example:

```
on doSomethingCritical
   global gDB, gTable
   gDB = New(Xtra "V12dbe", the MoviePath&"catalog.V12", "ReadWrite",
   "top top top")
   gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")
   -- other Lingo statements
   -- with a syntax error somewhere which causes the handler to abort
   gTable.mClose()
   gTable = 0
   gDB.mClose()
   gDB = 0
end doSomethingCritical
```

When a Lingo syntax error is detected and the handler aborts, you can still type

```
   gTable.mClose()
   gTable = 0
   gDB.mClose()
   gDB = 0
```

in the Message Window to close your database file.

Delivering to the End User

INM V12 Database is designed in a way that minimizes any last minute changes needed before delivery to the end-user. Unlike other database management systems where you need to swap the development version of certain files with the runtime versions, no swapping is required with INM V12 Database. (For security and usability reasons, Macromedia has developed a different process for locating and installing, over the Internet, for Shockwave applications – See Shockwave Projectors and Movies below).

Standalone Projectors

For an Xtra to be available to a Director projector, you must to place it in a folder named "Xtras" located in the same folder as the projector itself. The only files you need to deliver for INM V12 Database are **V12-Database.X32** (on Windows), **V12-Database.XTR** (on Mac OS X), and **V12-Database.XTR** (on Mac OS 9). If you are using Director on Mac OS X to publish projectors for both Mac OS X and OS 9, check the technote on our site called “How to publish projectors for cross-platform distribution”.

Alternately, you can package the required Xtras into your Director projector itself. Before creating the projector click Modify > Movie > Xtras window and select the INM V12 Database Xtra to include it with your projector. (See Auto-installation below for important details.)

As stated in the licensing agreement, you DO NOT deliver the license file "V12-30.LIC", which is in the System:Preferences folder of your Macintosh, or the Windows\System32 folder of your PC. It is not needed, nor is it recommended that you distribute this file to end users, due to the fact that it contains your personal licensing information.

Shockwave Projectors and Movies

Delivering a Shockwave Projector

A Shockwave Projector is very similar to a Director Projector, except that the file you deliver is very tiny; it contains no Xtras and no Director playback engine. See Auto-installation below for information on how to make sure that the appropriate v12 Database Xtra will download to the user's computer when needed.

To create a Shockwave Projector, select File > Create Projector > Options > Player > Shockwave.

Note: There are known problems with using external Director cast libraries in Shockwave on the Macintosh. Always test your projects thoroughly on both Macintosh and Windows platforms if you intend to deliver Shockwave applications.

Delivering a Shockwave Movie

The INM V12 Database Xtra cannot be included in Shockwave movies. Instead it is automatically downloaded to the end-user's computer, if required (i.e., if it is not already installed).

See Auto-installation below for information on how to make sure that the appropriate INM V12 Database Xtra will download to the user's computer when needed.

To test the resulting Shockwave movie locally on your computer, create a folder named DSWMEDIA anywhere on your hard disk and move the deliverable files into it (i.e., the Shockwave movie, the external castlibs, the HTML file, etc.)

Auto-installation

Whether you deliver your project as a Shockwave movie or projector, the INM V12 Database Xtras can be automatically installed to your end-users computers via the Internet. To instruct your movie to auto-install V12 when needed:

1 Make sure (or add) the following lines are in the "xtrainfo.txt" file located in your Director folder.
Important: Everything in square brackets must be on a single line without line breaks. Note that the corresponding Carbon version is automatically downloaded on Mac OS X, and the Classic version is downloaded on Mac OS 9.

```
; INM V12 Database for Director by Integration New Media, Inc.
; http://www.INM.com/
[#namePPC:"V12-Database.XTR", #nameW32:"V12-Database.X32",
   #package:"http://www.INM.com/downloads/products/v12director/verisi
   gn/3.4/v12",#info:"http://www.INM.com/products/v12director"]
```

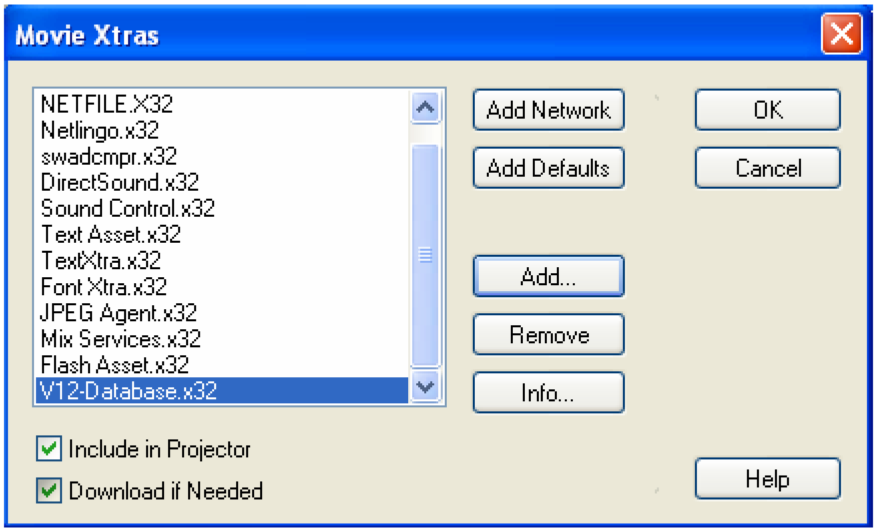
Note: The link to the Xtra's package follows the #package property. It contains the Xtra's version number (3.4 in the example).

You should copy the exact text from the Xtra's Release Notes and paste it into your "Xtrainfo.txt" file, as this information changes with every release.

2 Launch Director and open your movie.

3 Choose Modify > Movie > Xtras.

4 If "INM V12 Database for Director" is not in the list, click Add. Choose "V12-Database.X32" (Windows) or "V12-Database.XTR" (Mac) and click OK.



NETFILE.X32
Netlingo.x32
swadcmpr.x32
DirectSound.x32
Sound Control.x32
Text Asset.x32
TextXtra.x32
Font Xtra.x32
JPEG Agent.x32
Mix Services.x32
Flash Asset.x32
V12-Database.x32

[✓] Include in Projector
[✓] Download if Needed

5 Click "V12-Database.X32" (Windows) or "V12-Database.XTR" (Macintosh) in the Xtras list.

6 Make sure “Download if Needed” is checked.

7 Now, save your movie, or publish your Shockwave

At this point, Director tries to validate INM V12 Database Xtra's URL and reports any possible errors.

Users who access Shockwave movies that automatically download the INM V12 Database Xtra will get a Verisign certificate with the Xtra's credentials. Verisign is a standard means of downloading software from secure sources. Users have the choice of allowing or refusing the installation of the Xtras.

The appropriate Xtras for their operating system will be downloaded into a standard location on the hard drive, in a sub-folder of the Director Shockwave Player, and will then be accessible to any Shockwave movie thereafter.

Note: If the user refuses to download the Xtras, your application will not work correctly. You may want to use some defensive programming techniques to recover from such a situation.

Testing for end-users

It is always a good idea to thoroughly test the product before delivering it to the end-user. Tests must be performed on computers with configurations very similar to those of end-users. However, if you need to perform end-user tests on the computer that contains the INM V12 Database license, you can reproduce an end-user environment by proceeding as follows:

- Make sure Director or a Director Projector is not running,

- Open the System:Preferences folder of your Macintosh, or the (Windows\System folder) of your PC,
- Move the V12*.LIC file out of that folder to the destination of your choice, except of course, the trash can or the recycle bin,
- Open your project either with a Projector or Macromedia Director and perform the tests.
- Once the tests are completed, close the Projector or Macromedia Director and put the license file back into its original folder.

Note: DO NOT attempt to rename or tamper with the license file. If you do, you may need to re-register INM V12 Database.

Portability issues

Some files are cross-platform compatible and others are not. As a general rule, everything that can be executed is NOT cross-platform compatible. Applications, DLLs, Xtras, and EXE files are all executable and include the following:

- Macromedia Director
- Projectors generated by Macromedia Director
- the INM V12 Database Xtras

Everything that is a static document is cross-platform compatible. This includes:

- Director movies (either protected or not)
- INM V12 Database files

This figure identifies which files are specific to Macintosh or Windows, and which files are compatible with both Macintosh and Windows.

Windows only	**Macintosh and Windows**	**Macintosh only**
Director for Windows		Director for Macintosh
	Director movies (.DIR, .DXR) Shockwave movies (.DCR)	
Projector for Windows		Projector for Macintosh
INM V12 Database Xtras for Windows		INM V12 Database Xtras for Macintosh
	INM V12 Database files (.V12)	

Progress Indicators

INM V12 Database can display a progress indicator to the user when performing time-consuming tasks such as `mImport`, `mExportFile`, `mGetSelection`, `mSelect`, `mSelDelete`, `mFixDatabase` and `mPackDatabase`. Such a progress indicator can optionally feature a Cancel button to enable users to interrupt the current task. You can also replace the standard INM V12 Database progress bar by any custom progress indicator you provide via Director and Lingo.

To activate the progress indicator, set the `ProgressIndicator` property to `With_Cancel`, `Without_Cancel` or `UserDefined`. To deactivate it, set it to `None`.

Note: `mSelect` preceded by `mSetCriteria` with simple or complex criteria enables the display of a single progress indicator for the selection task, except if the criteria contain `wordStarts` or `wordEquals` operators. In that case, as many progress indicators as criteria are displayed.

Options of the ProgressIndicator Property

With_Cancel

INM V12 Database displays its own progress bar when performing one of the above-mentioned tasks. The user can click on the Cancel button to abort it. You can set the `ProgressIndicator.Message` property to whatever message you wish to display in the upper part of the progress window. If you set the `ProgressIndicator.Message` property to an empty string, INM V12 Database displays its own context-dependant message.

Without_Cancel

`Without_Cancel`: INM V12 Database displays its own progress bar when performing one of the above-mentioned tasks. No "Cancel" button is shown and the current task cannot be interrupted. You can set the `ProgressIndicator.Message` property to whatever message you wish to display in the upper part of the progress window. If you set the `ProgressIndicator.Message` property to an empty string, INM V12 Database displays its own context-dependant message.

UserDefined

INM V12 Database does not display a progress bar of its own. Instead, it calls three Lingo handlers that *must* be defined in your movie: `V12BeginProgress`, `V12Progress` and `V12EndProgress`. See User Defined Progress Indicators.

None

No Progress Indicator is shown and no callbacks are performed to Lingo handlers. *None* is the default value of the property.

See also Properties of Databases / ProgressIndicator.

User Defined Progress Indicators

If you wish to display your own progress indicator to the user, you must set the `ProgressIndicator` property to `UserDefined` and define three Lingo handlers in your movie: `V12BeginProgress`, `V12Progress` and `V12EndProgress`.

- The V12BeginProgress handler is called when the task starts, to allow you to initialize or open whatever progress indicator you want to show. One parameter is supplied to V12BeginProgress: it is either 100 (which is the upper bound that is eventually reached by the "progress" parameter, passed to the V12Progress handler, at the end of the operation), or -1 if no such upper bound is known up front.
- The V12Progress handler is repetitively called as long as the task is performed. Two parameters are supplied to V12Progress. The first parameter is the actual progress made so far and thus increases at every call. The second one is either 100 (which is the upper bound that is eventually reached by the first parameter at the end of the operation), or -1 if no such upper bound is known up front. V12Progress must return FALSE to keep INM V12 Database performing the current task or TRUE to abort it. See the example below.
- The V12EndProgress handler is called at the end of the task, to allow you to cleanup or close your progress indicator.

Warning: V12 methods must not be called from `V12BeginProgress`, `V12Progress` or `V12EndProgress`. This would create an infinite loop. Make sure to avoid this situation.

Example: spinning a custom cursor

If you want to spin a custom cursor while INM V12 Database is performing time-consuming tasks, you need to define the following three handlers:

```
on V12BeginProgress
   -- this is an empty handler. Spinning a cursor does not
   -- require intialization.
end V12BeginProgress

on V12Progress prog, limit
   -- rotate cursor cast members. limit is ignored because spinning
   -- a cursor does not need to reach an upper bound.
   -- cast 27=first of series 4 cursors, cast 31=mask (empty)
      cursor [27 + (prog mod 4), 31]
      return FALSE
end V12Progress

on V12EndProgress
      cursor -1  -- restore pointer cursor
```

```
            end V12EndProgress
```

In this example, V12BeginProgress is an empty handler, but it must be present. V12Progress only uses the prog parameter because it indefinitely rotates among four cursors (which are one-bit depth members defined in members 27, 28, 29 and 30). V12EndProgress is responsible for restoring the standard pointer cursor.

The "Progress" V12 Sample (available at: http://www.INM.com/support/v12director/technotes/) demonstrates several ways to implement progress indicators, including user-defined ones.

Properties of Databases

INM V12 Database files contain generic properties that provide for technical information on the current INM V12 Database environment (such as the number of available indexes and the state of the active debugger) and allow for the control of the INM V12 Database environment (such as custom string types and custom weekday names).

mSetProperty and mGetProperty are used to assign and read these generic database properties. Certain properties can only be read, not written (i.e. the number of available indexes) while others can be read and written (i.e. custom string types)

Certain properties are *persistent* (i.e. saved to the database and recovered when the database is reopened), others are not.

The syntax for mGetProperty is:

```
            val = mGetProperty(gDB, PropertyName)
```

The syntax for mSetProperty is:

```
            mSetProperty(gDB, PropertyName, Value)
```

PropertyName is a valid identifier (see Capacities and Limits for the definition of a valid identifier).

Value is always a string, even if PropertyName refers to a number.

Note: Value is limited to 4096 characters.

mSetProperty can be used to define a new property or to change an existing one. Using mSetProperty with a value of EMPTY deletes that property. Properties pertaining to Strings (see The String Property below) cannot be deleted.

Valid PropertyNames and Values are listed below. Both parameters must be of type String. Both are case insensitive (hence "resources", "Resources" and "RESOURCES" are all equivalent).

Note: mSetProperty is a *very* powerful tool. If you are unsure about what you're doing, always work on a copy of your original database.

You can retrieve the list of all the properties of a database by calling mGetPropertyNames, as in

```
            props = mGetPropertyNames(gDB)
```

Note: INM V12 Database properties can only be accessed by the `mSetProperty` and `mGetProperty` methods. They are totally unrelated to Windows file properties.

Predefined Properties

ProgressIndicator

ReadWrite, persistent. Valid values are "None", "With_Cancel", "Without_Cancel", "UserDefined". Default value is "None".

```
x = mGetProperty(gDB, "ProgressIndicator")
mSetProperty(gDB, "ProgressIndicator", "With_Cancel")
```

Enables INM V12 Database to show a progress indicator while performing time-consuming tasks, or calls back Lingo handlers to enable custom progress indicator implementations. See Progress Indicators.

ProgressIndicator.Message

ReadWrite, persistent.

```
msg = mGetProperty(gDB, "ProgressIndicator.Message")
mSetProperty(gDB, "ProgressIndicator.Message", "Exporting records.
   Please be patient…")
```

This property sets the text that is displayed in the upper part of INM V12 Database's progress window. If you set it to an empty string, INM V12 Database displays a message that depends on the current operation. See Progress Indicators.

VirtualCR

ReadWrite, persistent. Valid values: any ASCII character.

When importing or exporting data, convert Carriage Returns (ASCII #13) to this ASCII character. This is convenient to avoid the confusion of real Carriage Returns with Record Delimiters. This property can be overridden by a specific VirtualCR character passed as parameter to `mImport`.

Example:

```
c = mGetProperty(gDB, "VirtualCR")
put CharToNum(c) – show ASCII number in message window
--
mSetProperty(gDB, "VirtualCR", NumToChar(10)) -- define ASCII
   character #10 as virtual CR
```

See Step 2: Prepare the Data / Virtual carriage returns.

Note: The VirtualCR property requires an ASCII *character* as a parameter, not an ASCII number. To convert an ASCII number to a character, use the example provided here.

CharacterSet

ReadWrite, persistent. Valid values: "Windows-ANSI", "Mac-Standard", and "MS-DOS". Default: "Windows-ANSI" on the Windows version of INM V12 Database and "Mac-Standard" on the Macintosh version of INM V12 Database. This property affects all of

INM V12 Database's import and export functions. It can be overridden by a specific character set passed as a parameter to mImport.

Translates imported and exported files (whether Text or DBF) with the "Windows-ANSI", "Mac-Standard" or "MS-DOS" character set tables.

```
mSetProperty(gDB, "CharacterSet", "Mac-Standard")
```

See Step 2: Prepare the Data / Character Sets.

Resources

ReadOnly, non-persistent.

```
put mGetProperty(gDB, "resources")
```

Returns information on the number of available indexes and the index used by the last call to mSelect.

Example:

```
-- Number of indexes used: 6
-- Current index in table 'articles': 'nameNdx', using field 'name'
```

INM V12 Database resources should not be confused with the MacOS resources (those normally edited with ResEdit) - they are completely unrelated.

CurrentDate

ReadOnly, non-persistent.

mGetProperty returns the current date in INM V12 Database's raw format (YYYY/MM/DD) regardless of the Control Panel settings of the Mac or PC.

Example

```
aDate = mGetProperty (gDB, "CurrentDate")
put aDate
-- "2004/05/31"
```

Verbose

ReadWrite, non-persistent. Valid values are "on" and "off".

When Verbose is set to "on", INM V12 Database constantly displays a detailed feedback on the tasks it is performing in Director's Message Window,

Example:

```
mSetProperty(gDB, "verbose", "on")
mGetProperty(gDB, "verbose")
```

This property is extremely useful for debugging database errors, during testing. Turn verbose "on" at the beginning of a script segment that includes several INM V12 Database method calls; then turn it "off" at the end of that section of code. Although verbose is convenient for authoring and testing purposes, you should not use it in place of CheckV12Error to trap potential runtime errors and to display feedback appropriate for your application's end-users. See Errors and Defensive Programming.

Warning: If you put calls to set the verbose property on in your script, you must remember to remove them before distributing your application.

Months

ReadWrite, persistent. Valid values: any 12-word string.

The Month property contains the names of the months used by `mDataFormat` to format dates (the `MMMM` pattern in `mDataFormat`). The `Value` parameter is any 12-word string. Words must be separated by spaces. Names of months that contain spaces themselves must be enclosed between apostrophes.

Example:

```
mSetProperty (gDB, "Months", "Gennaio Febbraio Marzo Aprile Maggio
   Giugno Luglio Agosto Settembre Ottobre Novembre Dicembre")
```

ShortMonths

ReadWrite, persistent. Valid values: any 12-word string.

The ShortMonth property contains the short names of the months used by `mDataFormat` to format dates (the `MMM` pattern in `mDataFormat`). The `Value` parameter is any 12-word string. Words must be separated by spaces. Short names of months that contain spaces themselves must be enclosed between apostrophes.

Example:

```
mSetProperty (gDB, "ShortMonths", "Jan Fév Mar Avr Mai Juin Juil Août
   Sep Oct Nov Déc")
```

Weekdays

ReadWrite, persistent. Valid values: any 12-word string.

The Weekdays property contains the names of the weekdays used by `mDataFormat` to format dates (the `DDDD` pattern in `mDataFormat`). The `Value` parameter is any 12-word string. Words must be separated by spaces. Names of weekdays that contain spaces themselves must be enclosed between apostrophes.

Example

```
mSetProperty (gDB, "Weekdays", "Montag Dienstag Mittwoch Donnerstag
   Freitag Samstag Sonntag")
```

ShortWeekdays

ReadWrite, persistent. Valid values: any 12-word string.

The ShortWeekdays property contains the short names of the weekdays used by `mDataFormat` to format dates (the `DDD` pattern in `mDataFormat`). The `Value` parameter is any 12-word string. Words must be separated by spaces. Short names of weekdays that contain spaces themselves must be enclosed between apostrophes.

Example

```
mSetProperty (gDB, "ShortWeekdays", "Lun Mar Mie Jue Vie Sab Dom")
```

DBversion

ReadOnly, non-persistent. Returns the version of the INM V12 Database Xtra used to create the database.

Example:

```
v = mGetProperty(gDB, "DBversion")
put v
```

The above example puts "V12,3.4, Multi-User" in the message window.

FlushToDisk

If set to "true", this property will set the database to automatically flush the data to disk after every write operation.

V12 uses a disk cache system in order to speed up write operations, and normally flushes to disk on an irregular basis, as needed. This method is useful when you have an application running on a user's machine that may be prone to power failures, or when you want to be absolutely sure that you have stored a write operation to the hard drive.

If you prefer to manually control when the database is flushed to disk, use the mFlushToDisk method.

> **Tip:** A common use for the FlushToDisk property is in a kiosk situation where you store information into a database, and you want to increase the odds that a power-failure will not affect the database. Power failures may still cause problems with the operating system and/or database, but this command will ensure that the data has been written to the hard drive or storage device.

The String Property

The String property is covered in a separate section because other sub-properties (`Delimiters`, `StopWords` and `MinWordLength`) depend on it. The properties discussed below must be defined before fields of the corresponding string types are created in the database.

String.*Language*

ReadWrite, persistent. Valid values: any valid search/sort table.

The String property defines or modifies custom string types (i.e. string fields that obey specific searching and sorting rules). To define a new string type, or modify an existing one, you append its name to "String.". The chosen name must be a valid identifier and cannot contain periods (".").

Example:

```
mSetProperty (gDB, "String.Klingon", field "CompTable")
```

In this example, field "CompTable" contains the search/sort descriptor for Klingon. Once this property is defined, you can use the type "Klingon" to define new fields with `mCreateField` or `mReadDBstructure`. You also need to define this property first before modifying other string properties such as `Delimiters`, `StopWords` and `MinWordLength`.

To modify the sort order of the default string, just omit the *Language* identifier:

```
mSetProperty (gDB, "String", field "NewCompTable")
```

For details on various character sets and custom string types, please see Appendix 3: String and Custom String Types.

String.*Language*.Delimiters

ReadWrite, persistent. Valid values: any valid delimiters descriptor.

`Delimiters` defines, for an existing string type, the list of characters that are acceptable as word delimiters for full-text indexing. By default, word delimiters for the predefined types are all non-alphanumeric characters (everything except 0-9, A-Z, a-z and accented characters).

Example:

```
mSetProperty (gDB, "String.Spanish.Delimiters",
   "!?@$%?&*()[]^®{}£¢§¨¶≤≥º=+-,./\|")
```

In the above example, the punctuation characters indicated as the `Value` parameter are considered as delimiters.

If you need to specify the double-quote as part of the delimiters, either use the Lingo constant QUOTE, or place the delimiters in a Director member of type Field and use that field as a `Value` parameter as follows:

```
mSetProperty(gDB, "String.Spanish.Delimiters", field "myDelimiters")
```

All non-printing characters such as TAB, Space, CTRL+J, etc. (i.e. characters lower than ASCII 32) are always considered as delimiters.

To modify the delimiters of the default string, just omit the *Language* identifier:

```
mSetProperty(gDB, "String.Delimiters", field "newDelimiters")
```

String.*Language*.MinWordLength

ReadWrite, persistent. Valid values: an integer in the range 2..8 passed as a String parameter.

MinWordLength determines the size of the shortest word that must be considered for full-indexing. All words shorter than MinWordLength are ignored and hence refused by the mSetCriteria method when used with the operator "WordEquals".

Example:

```
mSetProperty (gDB, "String.Spanish.MinWordLength", "3")
```

Note that the `Value` parameter is "3" (with quotation marks). This is because mSetProperty expects a `Value` parameter of type String only. The following is also a valid formulation:

```
mSetProperty(gDB, "String.Spanish.MinWordLength", String(3) )
```

To modify the `MinWordLength` of the default string, just omit the *Language* identifier:

```
mSetProperty(gDB, "String.MinWordLength", "2")
```

The default value for MinWordLength is 4. The MinWordLength property must be defined *before* creating fields of type String, in the database, so that the full-index is created accordingly.

String.*Language*.StopWords

ReadWrite, persistent. Valid values: a string no longer than 32K.

StopWords allows for the definition of a list of words that must be ignored in the full-indexing process. The `Value` parameter is a string containing the stop words in any order separated by spaces, TABs or Carriage Returns).

Example:

```
mSetProperty (gDB, "String.Spanish.StopWords", "el está en la de ")
```

To modify the stop words list of the default string, just omit the *Language* identifier:

```
mSetProperty(gDB, "String.StopWords","a the on for in by as")
```

By default, the StopWords property is empty. A typical list of stop words in English is:

```
“a by in the an for is this and from it to are had not was as have
of with at he on which be her or you but his that”
```

Note: Remember that at most 4096 characters can be stored in individual properties.

Custom Properties (Advanced Users)

Advanced users may want to define their own properties and make them persistent to a database. This is a convenient way to store preferences in your database and it eliminates the trouble of having to create a table that includes only one record.

For example, if you need to save the frame last visited by the user prior to leaving your application (possibly to bring him/her back to that same frame next time):

```
on StopMovie
  global gDB
  mSetProperty(gDB, "LastVisited", string(the frame))
  -- Typcasting to string is mandatory here.
  -- Some other housekeeping tasks...
end StopMovie
```

The startup handler of your movie would contain:

```
on StartMovie
  global gDB
  -- create database instance, etc.
  myLastVisit = integer(mGetProperty(gDB, "LastVisited"))
  if (myLastVisit)>0) then go to frame myLastVisit
  -- other housekeeping tasks...
end StartMovie
```

Custom properties are always read-write and persistent.

Capacities and Limits

Database

- The size of an INM V12 Database file is limited by disk space. See Table Index and Field capacities for limitations when using the Light Edition of INM V12 Database.
- The number of database instances and table instances is limited by RAM. Each executable is entitled to only one database instance of a given database (although, you may create as many instances of *distinct* databases as you wish in a single executable). Multiple instances of a table can be created on a single computer. Note that you may run into significant performance issues as the number of open databases increases.
- A valid INM V12 Database file contains at least one table, and each table requires at least one field and one index.

Note: Previous versions of INM V12 Database supported a **Shared ReadWrite** mode for multi-user access to V12 Databases over a network. If you wish to use this functionality we can supply you with the necessary information to do so. However, this feature is no longer supported.

Creation

- In order to create licensed V12 database structures at runtime, you must have a registered INM V12 Database license file on your workstation. If you do not have a licensed INM V12 Database on the workstation, your databases will be unlicensed and will display a splash screen each time they are opened. The best approach is often to clone new databases from existing databases.

Selection

- With INM V12 Database Regular Edition, up to 100 criteria can be chained with sequences of mSetCriteria separated by Boolean operators.
- With INM V12 Database Light Edition, up to 3 criteria can be chained with sequences of mSetCriteria separated by Boolean operators.

Importing

- DBF files of type DBase III, DBase IV, DBase V, FoxPro 2.0, FoxPro 2.5, FoxPro 2.6, FoxPro 3.0 and FoxPro 5.0 can be imported, exported and used as templates for the definition of V12 databases. Fields of type DateTime are not supported. The following DBF data types are ignored: General, Character-Binary, Memo-Binary.
- On Windows, MS Access databases, MS FoxPro files, MS Excel workbooks and MS SQL Server data sources can be used as templates

for creating new V12 databases and as sources of data for importing records through ODBC drivers. The exact database translation/data importing rules varies among ODBC drivers and versions of ODBC drivers.

Table

- The maximum number of tables in an INM V12 Database file, using INM V12 Database Regular Edition is 128.
- Only one table can be handled by the Light Edition of INM V12 Database.
- Identifiers (names of fields, tables and indexes) are limited to 32 characters. They must start with a low-ASCII alphabetic character (a..z, A..Z), which can be followed by any alphanumeric character (0..9, a..z, A..Z, à, é, ö). Keywords such as `NOT`, `AND`, `OR`, `String`, `Integer`, `Float`, `Date` or `Media` are not suitable for use as identifiers.

Field

- With the Regular Edition of INM V12 Database there is no limit on the number of fields permitted in a database.
- With the Light Edition of INM V12 Database, up to 16 fields can be defined (in a single table).
- No two fields or indexes can have the same name in the same table. However, two fields or two indexes might have the same name in different tables.
- All records are of variable length. Fields of type `media` are limited to 1Mb. Fields of type `string` are limited to 64K.
- The range of the type `Integer` is -2^{31} to $2^{31}-1$ (-2147483648 to 2147483647).
- The range of the type `Float` is ±1.79769313486232E+308 to ±2.22507385850720E-308.
- Any date later than January 1st, 1600 can be compared, retrieved and stored to fields of type `Date`. However, date formatting is limited to the range Jan 1st 1904 through Dec 31st 2037.
- Fields of type `media` can hold almost any type of media that can be stored in a Director member, such as vector shapes, animated GIFs, Flash movies, sound files, etc. The known exceptions are Film Loops and QuickTime movies.

Index

- A maximum of 128 indexes can be defined on an INM V12 Database file using the Regular Edition of INM V12 Database. Each index can operate on up to 12 fields.
- With the Light Edition of INM V12 Database, A maximum of 12 indexes can be defined on a V12 Database file and only one field can be use by an index.
- Up to 32 custom string types can be defined.

- When indexing fields of type String, up to the 251 first characters of each string are actually entered in the index. The remaining characters are ignored.
- Full-indexes are built with words not exceeding 31 characters. Words longer than 31 characters are truncated to 31 characters for the purpose of indexing (this does not affect the actual data).

V12 Database Methods

Initialization and closing
new (DB)
new (T)
mClose (DB)
mClose (T)

Retrieving Data
mGetField (T)
mGetMedia (T)
mGetSelection (T)
mGetUnique (T)
mDataFormat (T)

Modifying Data
mAddRecord (T)
mDeleteRecord (T)
mEditRecord (T)
mSelDelete (T)
mSetField (T)
mSetMedia (T)
mUpdateRecord (T)

Browsing through Data
mFind (T)
mGetPosition (T)
mSelectCount (T)
mGo (T)
mGoFirst (T)
mGoLast (T)
mGoNext (T)
mGoPrevious (T)

Searching and Sorting
mOrderBy (T)
mSelect (T)
mSelectAll (T)
mSetCriteria (T)
mSetIndex (T)

Importing\ Exporting Data
mImport (T)
mExportSelection (T)

Error Management
V12Error (G)
V12Status (G)

Database Structures
mBuild (DB)
mCloneDatabase (DB)
mDumpStructure (DB)
mReadDBStructure (DB)

Database Utilities
mFixDatabase (S)
mGetProperty (DB)
mGetPropertyNames (DB)
mNeedSelect (T)
mSetProperty (DB)
mSetPassword (DB)

Network Utilities
V12Download (G)
V12DownloadInfo (G)
mXtraVersion (S)

G = Global Method
DB = Database Method
T = Table Method
S = Static Method

mAddRecord (T)

Syntax

`mAddRecord(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Adds a new record to table *whichTable* and sets it as the current record.

Calls to mAddRecord are generally followed by calls to mSetField or mSetMedia and must end with a call to mUpdateRecord. If you call any method other than mSetField or mSetMedia before calling mUpdateRecord, no record is added to *whichTable*.

If the database to which *whichTable* belongs is open in "ReadOnly" mode, mAddRecord signals an error. Errors can be retrieved with V12Error or V12Status.

Example

This script adds a record, puts the value "funnel" in field "name", the value 2.95 in field "price" and updates the record:

```
mAddRecord(gTable)
CheckV12Error() -- this handler needs to be defined by you
mSetField(gTable, "name", "funnel")
CheckV12Error()
mSetField(gTable, "price", 2.95)
CheckV12Error()
mUpdateRecord(gTable)
CheckV12Error()
```

See Also

`mEditRecord (T), mSetField (T), mSetMedia (T),mUpdateRecord (T)`

mBuild (DB)

Syntax

mBuild(*whichDatabase [, onlineDB]*)

Parameters

whichDatabase = Instance of a V12 Database.
onlineDB = Pass "Online" to create a V12Online compatible database. Optional parameter.

Description

Verifies the validity of the database structure defined by `mReadDBstructure` and writes the database structure to the file.

After successfully calling `mBuild`, the database remains open in "ReadWrite" mode. Data can be immediately imported to it. It is very important to check for possible errors via `V12Error` or `V12Status` after calling `mBuild`.

Example

These statements read the database definition contained in the Director cast member of type Field named "DBstruct" and create a V12 database named "myBase.V12" with password "very secret password":

```
set gDB = New(Xtra "V12DBE", the moviePath & "myBase.V12", "Create", "very secret
password")
CheckV12Error() -- this handler needs to be defined by you
mReadDBStructure(gDB, "Literal", field "DBstruct")
CheckV12Error()
mBuild(gDB)
CheckV12Error()
```

See Also

`new (DB), mReadDBStructure (DB)`

mCloneDatabase (DB)

Syntax

```
mCloneDataBase(whichDatabase, pathName)
```

Parameters

whichDatabase = Instance of INM V12 Database.
pathName = Pathname of the new V12 Database file.

Description

Create a clone of the V12 database referred by *whichDatabase* (an identical copy, with the same structure but no records).

Example

```
mCloneDataBase (gDB, The MoviePath & "Clone.V12")
CheckV12Error() -- this handler needs to be defined by you
```

See Also

```
new (DB)
```

mDataFormat (T)

Syntax

```
mDataFormat(whichTable, fieldName, format)
```

Parameters

whichTable = Instance of a V12 Table.
fieldName = The name of a field in *whichTable.*
format = The data format that will be applied to *fieldName*.

Description

Associates a formatting pattern to the data retrieved from *fieldName*. Data formats can be applied to fields of type Float, Integer and Date. If the retrieved data are longer than the formatting pattern, V12 returns the corresponding number of "#"s (see Example 1).

When a table is first opened, the default format of all its Date fields is set to "yyyy/mm/dd". Float and Integer fields return as many digits as required by the data.

Valid patterns for fields of type Integer and Float contain the following:

9	A zero at that position if no digit is present
#	A space at that position if no digit is present
. (period)	decimal point
other	other characters are literally reproduced.

Valid patterns for fields of type Date are combinations of:

D	day
M	month
Y	year
other	other characters are literally reproduced.

By default, the names for the months in INM V12 Database are (MMMM)

```
January, February, March, April, May, June, July, August, September, October,
November, December
```

The short names for the months are (MMM)

```
Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
```

The names for the weekdays are (DDDD)

```
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
```

The short names of the weekdays are (DDD)

```
Sun, Mon, Tue, Wed, Thu, Fri, Sat
```

All of these names can be replaced by custom names through the properties Months, ShortMonths, Weekdays and ShortWeekdays (see `mGetProperty` and `mSetProperty`)

Example 1

The pattern "###9999" forces the output of an integer field to be formatted within no less than four digits and with three leading spaces if necessary. Thus:

```
4           is formatted as    "   0004"
123         is formatted as    "   0123"
314159      is formatted as    " 314159"
3141592     is formatted as    "3141592"
31415926    is formatted as    "#######"
```

Example 2

This example forces the output of the field `ratio` to 2 integral digits, 2 decimal places and a trailing "%" sign. If the value in field `ratio` is 12.3489, the output is formatted as "12.35%"

```
mDataFormat(gTable, "ratio", "99.99%")
put mGetField(gTable, "ratio")
```

Example 3

The pattern "(999) 999-9999" can format phone numbers stored as integers. The following statements return a string formatted as "(514) 871-1333".

```
mDataFormat(gTable, "phone", "(999) 999-9999")
put mGetField(gTable, "phone")
```

Example 4

The following statements retrieve the formatted content of field "Birthday" of the current record and put it in the Director field "BD". The result is formatted as "29/01/95".

```
mDataFormat(gTable,"Birthday","dd/mm/yy")
put mGetField(gTable, "Birthday") into field "BD"
```

Example 5

The pattern	*Formats 29 January 1995 as*
D	29
DDDD	Thursday
MM	01
DD-MM	29-01
MMM DD, YY	Jan 29, 95
On D MMMM, YYYY	On 29 January, 1995

See Also

```
mGetField (T), mGetSelection (T), mGetProperty (DB), mSetProperty (DB)
```

mDeleteRecord (T)

Syntax

```
mDeleteRecord(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Deletes the current record from table *whichTable* .

After calling `mDeleteRecord`, the record following the current record becomes the new current record. If no record follows the deleted record, the preceding record becomes the new current record. If no record precedes the deleted record, the current record is not defined.

The database must be open in "ReadWrite" or "Shared ReadWrite" mode for `mDeleteRecord` to succeed. Otherwise, `mDeleteRecord` fails (no record is deleted) and an error is signaled by `V12Error` and `V12Status`.

Example

```
mDeleteRecord(gTable)
```

See Also

```
mSelDelete (T)
```

mDumpStructure (DB)

Syntax

```
mDumpStructure(whichDatabase, dumpMode*)
```

Parameters

whichDatabase = Instance of a V12 Database.
dumpMode = What information is to be retrieved about *whichDatabase*'s structure
** Some dumpModes can require more parameters*

Description

Retrieves information on the structure of database *whichDatabase*. `mDumpStructure`'s output is typically put in the Message Window or in a field member. `mDumpStructure` is very convenient for debugging.

The possibilities for *dumpMode* are:

extended: Returns each table of your database with all fields by type, size and index information.

```
put mDumpStructure(gdb, "Extended")
```

formal: Returns the standard format used in creating a database.

```
put mDumpStructure(gdb, "Formal")
```

tableCount: Returns the number of tables in the database.

```
put mDumpStructure(gdb, "tableCount")
```

tableNames: Returns the names of all tables in the database.

```
put mDumpStructure(gdb, "tableNames")
```

fieldCount: Outputs the number of fields in a specific table.

```
put mDumpStructure(gdb, "FieldCount", tableName)
```

fieldNames: Returns the names of the fields in a specific `table`.

```
put mDumpStructure(gdb, "FieldNames", tableName)
```

fieldType: Returns the type of a field in a specific table.

```
put mDumpStructure(gdb, "fieldType", tableName, fieldName)
```

fieldSize: Returns the size of a field in a specific table (field of type string or media only).

```
put mDumpStructure(gdb, "fieldSize", tableName, fieldName)
```

indexCount: Returns the number of indexes in a specific table.

```
put mDumpStructure(gdb, "indexCount", tableName)
```

indexNames: Returns the indexed fields in a specific table.

```
put mDumpStructure(gdb, "indexNames", tableName)
```

full-Indexes: Returns the fully indexed fields in a specific `table`.

```
put mDumpStructure(gdb, "full-indexes", tableName)
```

defaultIndex: Returns the default index for a specific table

```
put mDumpStructure(gdb, "defaultIndex", tableName)
```

Example

```
The following line puts the structure of your database into the message window
put mDumpStructure(gDB)


The following line puts the structure of your database into field dummy using
-- the Extended dumpMode.
put mDumpStructure(gDB, "Extended") into field "dummy"
```

mExportSelection (T)

Syntax

mExportSelection(*whichTable*, *exportType*, *pathName* [, *fieldDelimiter* [, *recordDelimiter* [, *fieldName*]*]])

Parameters

whichTable = Instance of a V12 Database.
exportType = "TEXT" or "DBF"
pathName = Pathname of exported file.
fieldDelimiter = One or multiple character delimiters (valid for "TEXT" *exportType* only). Optional parameter.
recordDelimiter = One or multiple character delimiters (valid for "TEXT" *exportType* only). Optional parameter.
*fieldName** = Name(s) of field(s) to export, in that order. Optional parameter.

Description

Export the selection as currently sorted taking each field's `mDataFormat` setting and the CharacterSet property into consideration. To export the whole table, make sure to call `mSelectAll` first. The first line of the exported file contains the names of the exported fields (to remain compatible with `mImport`).

Example

```
-- Export the data of all the fields of the current table:
put mExportSelection(gTable, "TEXT", the moviePath & "selection.txt", TAB, RETURN)


-- Export data of field 'price' of the current table:
```

```
put mExportSelection(gTable, "TEXT", the moviePath & "selection.txt", TAB, RETURN,
"price")

-- Export in the file "Export.dbf" the data of all fields of the current table:
mExportSelection(gTable, "DBF", the moviePath & "Export.dbf")

-- Export in the file "Export.dbf" the data of fields field1 and field2:
mExportSelection(gTable, "DBF", the moviePath & "Export.dbf", "field1", "field2")
```

See Also

mDataFormat (T), mSetProperty (DB), mImport (T)

mEditRecord (T)

Syntax

`mEditRecord(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Enables the modification of the current record.

A normal record modification sequence consists of a call to `mEditRecord`, a sequence of calls to `mSetField or mSetMedia,` followed by a call to `mUpdateRecord`.

If you call any method other than `mSetField or mSetMedia` before calling `mUpdateRecord`, the record modification process is aborted.

If there is no current record, or if the database to which *whichTable* belongs is open in "ReadOnly" mode, `mEditRecord` signals an error that can be checked with `V12Error` or `V12Status`.

Example

The following script enables the modification of the current record. It then puts the value "funnel" in field "name", the value 2.95 in field "price" and updates the record:

```
mEditRecord(gTable)

CheckV12Error() -- this handler needs to be defined by you
mSetField(gTable, "name", "funnel")

CheckV12Error()
mSetField(gTable, "price", 2.95)

CheckV12Error()
mUpdateRecord(gTable)

CheckV12Error()
```

See Also

mSetField (T), mSetMedia (T), mUpdateRecord (T)

mFind (T)

Syntax

`mFind(`*`whichTable`*`, `*`direction`*`, `*`keyword`*`)`

Parameters

whichTable = Instance of a V12 Table.
direction = "FIRST", "NEXT" or "PREVIOUS"
keyword = What you want to search for. This is used only when the *direction* parameter is “First”.

Description

Sets the current record to one, in *whichTable*'s selection, whose Master Field equals or starts with *keyword*. The Master Field is the one that determines the sorting order of the selection.

When the "First" parameter is used, the first matching occurrence is found. Any sequence of calls to `mFind` must first start with one that uses the "First" parameter and includes the *keyword*.

When the "Next" parameter is used, the record following the current matching record is found.

When the "Previous" parameter is used, the record preceding the current matching record is found.

If no records match *keyword*, the current record remains unchanged and an error is signaled. It can be checked with `V12Error` or `V12Status`.

Example 1

The following statement sets the current record to the first one in the selection that starts with the letter "s":

```
mFind(gTable, "First", "S")
```

Example 2

The following statements output up to the first five occurrences of "Gonzales" without affecting the selection (ie, the same could be achieved using `mSetCriteria` / `mSelect`, to the cost of modifying the selection):

```
mFind(gTable, "First", "Gonzales")
CheckV12Error() -- this handler needs to be defined by you
if V12Status() then exit
put mGetField(gTable, "name")
CheckV12Error()

Set flag = TRUE
repeat while flag = TRUE
   mFind(gTable, "Next")
   if V12Status() then
           set flag = FALSE
   else
           put mGetField(gTable, "name")
       CheckV12Error()
   End if
end repeat
```

See Also

```
mGo (T), mGoFirst (T), mGoLast (T), mGoPrevious (T),
mGoNext (T), mSetCriteria (T), mOrderBy (T), mSelect (T), mSelectAll (T)
```

mFixDatabase (S)

Syntax

```
mFixDatabase(Xtra "V12DBE", databaseName, newDatabaseName)
```

Parameters

databaseName = The pathname to the file you want to fix.
newDatabaseName = The new pathname of the fixed file.

Description

Fixes a corrupted database. To be fixed, a database must first be closed. mFixDatabase is a static method (ie, it operates on Xtra "V12DBE", not an instance of Xtra "V12DBE")

Example

```
mFixDatabase(Xtra "V12DBE", the moviePath & "DatabaseName.v12", the moviePath &
"FixedDatabase.V12")
```

See Also

mPackDatabase (DB)**Error! Reference source not found.**

mFlushToDisk (DB)

Syntax

mFlushToDisk(*whichDatabase*)

Parameters

whichDatabase = Instance of a V12 Database.

Description

Flushes to disk what is presently in the users cache. This method is useful when you have an application running on a user's machine that may be prone to power failures, or when you want to be absolutely sure that you have stored a write operation to the hard drive. V12 uses a disk cache system in order to speed up operations, and normally flushes to disk on an irregular basis, as needed.

This method should be used when you are constantly writing to a V12 database.

Note: Flushing to Disk can be performed automatically by using the `mSetProperty` command to set the "FlushToDisk" property to True. That property will set the database to automatically flush the data to disk after every write operation. If you set the FlushToDisk property to True, then the `mFlushToDisk` method is not necessary. Flushing to disk is only useful in "ReadWrite" databases. Also, if this property is set to TRUE, it will take additional CPU time and can slow down database response.

Example

```
repeat with i = 1 to 1000
        mAddRecord(gTable)
        CheckV12Error() -- this handler needs to be defined by you
        mSetField(gTable, "UniqueID", i)
        CheckV12Error()
        mUpdateRecord(gTable)
        CheckV12Error()
    end repeat
```

```
mFlushToDisk(gdb)
```

See Also

```
mSetProperty (DB)
```

mGetField (T)

Syntax

```
mGetField(whichTable, fieldName [, dataFormat])
```

Parameters

whichTable = Instance of a V12 Table.
fieldName = Name of the field to read in *whichTable.*
dataFormat = Pattern for Integer, Float and Date formatting. Optional parameter.

Description

Retrieves the content of field *fieldName* of the current record.

If the current record is not defined (i.e., the selection is empty), `mGetField` returns <Void>.

If *`dataFormat`* is specified, the retrieved data is formatted accordingly (see `mDataFormat` for a list of valid patterns). Otherwise, if a formatting pattern is assigned to *fieldName* with `mDataFormat`, that format is used. `mGetField`'s *dataFormat* parameter overrides `mDataFormat`'s setting.

Example

```
set name = mGetField(gTable, "theName")
CheckV12Error() -- this handler needs to be defined by you
set date = mGetField(gTable, "theDate", "YY-MM-DD")
CheckV12Error()
```

See Also

```
mDataFormat (T), mGetSelection (T), mGetMedia (T), mSetField (T)
```

mGetMedia (T)

Syntax

```
mGetMedia(whichTable, fieldName, memberRef)
```

Parameters

whichTable = Instance of a V12 Table.
fieldName = Name of the field to read in *whichTable*.
memberRef = The Director member that will store the retrieved media.

Description

Retrieves the media contained in field *fieldName* of the current record and stores it in *memberRef* .

If *fieldName* is a V12 field of type String, `mGetMedia` attempts to retrieve the styled text associated with it. If no styled text was stored in *fieldName*, *memberRef* is emptied.

Examples

```
--The following lines are equivalent; they retrieve the media
--from field "Photo" in current record, and store it in cast member "Image"
```

```
mGetMedia(gTable, "photo", member "Image")
mGetMedia(gTable, "photo", member "Image" of castlib 1)
```

See Also

`mGetSelection (T), mGetField (T), mSetMedia (T), mSetField (T)`

mGetPosition (T)

Syntax

`mGetPosition(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Returns the position of the current record in the selection. If the selection is empty, `mGetPosition` returns 0.

Example

```
set p = mGetPosition(gTable)
CheckV12Error() -- this handler needs to be defined by you
set n = mSelectCount(gTable)
CheckV12Error()
put "Showing record " & p & " out of " & n
```

See Also

```
mGo (T), mGoPrevious (T), mGoNext (T), mGoFirst (T),
mGoLast (T), mSelectCount (T)
```

mGetPropertyNames (DB)

Syntax

`mGetPropertyNames(`*`whichDatabase`*`)`

Parameters

whichDatabase = Instance of a V12 Database.

Description

Returns the names of all the properties defined. The result is a string with one property name per line.

Example

```
put mGetPropertyNames(gDB)
```

Returns the following text in the Message Window with any Custom properties:

```
creation-date
DBE-NAME
dbversion
LogError
MaxLoggedErrors
modification-date
months
myOwnProp
ProgressIndicator
```

```
shortmonths
shortweekdays
tbl$lastmodified
weekdays
```

See Also

mSetProperty (DB), mGetProperty (DB)

mGetProperty (DB)

Syntax

mGetProperty(*whichDatabase*, *propertyName*)

Parameters

whichDatabase = Instance of a V12 Database.
propertyName = The property in *whichDatabase*.

Description

Retrieves the value of property *propertyName*. The result is a string.

Possible values of *propertyName* are custom properties or:

ProgressIndicator

ProgressIndicator.Message

VirtualCR

CharacterSet

Resources

CurrentDate

Verbose

Months

ShortMonths

Weekdays

ShortWeekdays

ErrorLog

MaxLoggedErrors

SharedRWcount

DBversion

String.*Language*

String.*Language*.Delimiters

String.*Language*.MinWordLength

String.*Language*.StopWords

FlushToDisk

Example

This statement returns the character set to be used when importing and exporting records:

```
put mGetProperty(gDB, "characterSet")
```

Example

This statement assigns the string "January February March April May June July August September October November" to the variable *theMonths.*

```
set theMonths = mGetProperty(gDB, "months")
```

See Also

`mSetProperty (DB)`

mGetSelection (T)

Syntax

```
mGetSelection(whichTable, [outputType [, from [, #recs [, fieldDelimiter] [,
        recordDelimiter] [, fieldNames ]* ]]])
```

Parameters

whichTable = Instance of a V12 Table.
outputType = "LITERAL", "LIST" or "PROPERTYLIST". Optional parameter.
from = The first record to be retrieved. Optional parameter.
#recs = The number of records to retrieve. Optional parameter.
fieldDelimiter = The field delimiter of the records returned. Optional parameter.
recordDelimiter = The record delimiter of the records returned. Optional parameter.
fieldNames = The fieldname(s) in *whichTable* to be returned. Optional parameter.

Description

Retrieves one or more fields, within one or more records of the selection in *whichTable*.

`mGetSelection` is convenient for populating scrolling lists or popup menus in the user interface.

If *outputType* is omitted, "LITERAL" is assumed.
"LITERAL" returns a string formatted as specified by *fieldDelimiter* and *recordDelimiter*.
"LIST" returns a Lingo list containing as many lists as records in the selection (a list of lists).
"PROPERTYLIST" returns a Lingo property list.

If the selection contains one record, the "LIST" and "PROPERTYLIST" output types return a list containing one list.

If the selection is empty, `mGetSelection` returns an empty string ("LITERAL") or an empty list ("LIST" or "PROPERTYLIST").

from is the number of the first record to retrieve. If omitted, the first record in the selection is used.

#recs is the number of records to retrieve. It is optional. If omitted, all records up until the end of the selection are retrieved.

fieldDelimiter is the delimiter to insert between each field. It can only be used with the "LITERAL" output type. Even then, it is optional. If omitted, TAB is used.

recordDelimiter is the delimiter to insert at the end of each record. It can only be used with the "LITERAL" output type. Even then, it is optional. If omitted, RETURN is used.

fieldNames are as many field names as required in the output. Invalid field names are ignored. If omitted, all fields in the current table are returned.

Call `V12Status` or `V12Error` to check if `mGetSelection` succeeded.

Example 1

This statement retrieves the entire selection to a Lingo list and places it in a variable named myList. All fields and all records are retrieved for further operation on the variable myList.

```
set myList = mGetSelection(gTable, "LIST")
```

Example 2

This statement retrieves the values of the fields LastName and FirstName of the entire selection to the field PickList. LastName and FirstName are separated by a comma.

```
mSelectAll(gTable)
-- mSelectCount(gTable) is used to know exactly how many records are returned
put mGetSelection(gTable, "LITERAL", 1, mSelectCount(gTable), ", ", RETURN,
"LastName", "FirstName") into field "PickList"
```

See Also

```
mGetField (T), mGetUnique (T), mSetField (T), mDataFormat (T)
```

mGetUnique (T)

Syntax

`mGetUnique(`*`whichTable`*`, `*`outputType`*`)`

Parameters

whichTable = Instance of a V12 Table.
outputType = "LITERAL" or "LIST"

Description

Retrieves unique values from the master field. The master field is the one that determines the selection's sorting order.

`mGetUnique` is convenient for populating scrolling lists or popup menus with unique values of the sorted field of a selection.

`outputType` is either "LITERAL" or "LIST". It is mandatory. "LITERAL" retrieves a string with one value per line. "LIST" retrieves a list of single-item lists (for consistency with `mGetSelection`).

If the selection is empty, `mGetUnique` returns an empty string ("LITERAL") or an empty list ("LIST").

Example

These statements retrieve the unique values in field "color" and place them in field "PopMenu". It is important to call `mOrderBy` in order to set "color" as the master field. For faster processing, "color" must be an indexed field.

```
mSetCriteria(gTable, "model", "=", "python")
CheckV12Error() -- this handler needs to be defined by you
mOrderBy(gTable, "color")
CheckV12Error()
mSelect(gTable)
CheckV12Error()
if (mSelectCount(gTable) = 0) then
       alert "No item matches your choice"
```

```
else
        put mGetUnique(gTable, "literal") into field "PopMenu"
end if
```

See Also

`mSetField (T), mDataFormat (T), mSetCriteria (T), mOrderBy (T), mSelect (T)`

mGo (T)

Syntax

`mGo(`*`whichTable`*`, `*`toPosition`*`)`

Parameters

whichTable = Instance of a V12 Table.
toPosition = The record number to go to in *whichTable* .

Description

Sets the current record to the *toPosition*th record of the selection.

If *toPosition* is not within the selection's range, the statement is ignored and the position of the current record remains unchanged. `V12Error` and `V12Status` report an error code.

Example

These statements set the current record to the selected line in a list of records previously retrieved with mGetSelection:

```
set userChoice = the mouseLine
if (userChoice > 1) then
    mGo(gTable, userChoice)
    UpdateDisplay()
end if
```

See Also

`mGoFirst (T), mGoLast (T), mGoNext (T), mGoPrevious (T), mGetPosition (T), mSelectCount (T)`

mGoFirst (T)

Syntax

`mGoFirst(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Sets the current record to the first record of the selection.

Example

```
mGoFirst(gTable)
```

See Also

`mGoLast (T), mGoPrevious (T), mGoNext (T), mGo (T), mGetPosition (T), mSelectCount (T)`

mGoLast (T)

Syntax

```
mGoLast(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Sets the current record to the last record of the selection.

Example

```
mGoLast(gTable)
```

See Also

```
mGoFirst (T), mGoPrevious (T), mGoNext (T), mGo (T), mGetPosition (T), mSelectCount
(T)
```

mGoNext (T)

Syntax

```
mGoNext(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Sets the current record to the record following the actual current record in the selection.

If the current record is already the last record of the selection, `mGoNext` is ignored and an error code is signaled by `V12Error` and `V12Status`.

Example

```
mGoNext(gTable)
```

See Also

```
mGoFirst (T), mGoLast (T), mGoPrevious (T), mGo (T), mGetPosition (T), mSelectCount
(T)
```

mGoPrevious (T)

Syntax

```
mGoPrevious(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Sets the current record to the record preceding the actual current record in the selection.

If the current record is already the first one in the selection, `mGoPrevious` is ignored and an error code is signaled by `V12Error` and `V12Status`.

Example

```
mGoPrevious(gTable)
```

See Also

```
mGoFirst (T), mGoLast (T), mGoNext (T), mGo (T), mGetPosition (T), mSelectCount (T)
```

mImport (T)

Syntax

```
mImport(gTable, "Text", fileToImport [, optionsList])
mImport(gTable, "Literal", stringToImport [, optionsList])
mImport(gTable, "DBF", fileToImport [, optionsList])
mImport(gTable, "List", listToImport)
mImport(gTable, "PropertyList", listToImport)
mImport(gTable, "V12", fileToImport, password, tableName)
mImport(gTable, "Access", fileToImport, userName, password, tableName) -- Win only
mImport(gTable, "FoxPro", path) -- Windows only
mImport(gTable, "Excel", fileToImport, tableName) -- Windows only
mImport(gTable, "SQLserver", DSN, userName, password, tableName) -- Windows only
```

Description

Imports data from a file or data source into the current table. Possible input types are:

- "Text": A text file (*.txt).
- "Literal": A Director field, text member or expression.
- "DBF": A dbf file (*.dbf).
- "List": A Lingo list.
- "PropertyList": A Lingo property list.
- "V12": A V12 database (*.v12).
- "Access": An MS Access file (*.mdb).
- "FoxPro": An MS FoxPro file (*.dbf).
- "Excel": An MS Excel file (*.xls).
- "SQLserver": An MS-SQL Data Source Name (DSN).

Input type	Parameters
TEXT	*fileToImport* = Pathname of the text file to import. The file's first line must contain field descriptors. *optionsList* = Lingo property list containing the following properties (all optional): #FieldDelimiter character. Default: TAB. #RecordDelimiter character. Default: RETURN. #CharacterSet "Windows-ANSI", "Mac-Standard" or "MS-DOS". Default: Matches current OS. #VirtualCR character: Default: NumToChar(11). #TextQualifier character. Default: "|"

Examples:
This example imports a text file with the default options:
mImport(gTable, "Text", the moviePath & "test.txt")
This example imports a text file with hyphens as field delimiter and ASCII #14 as record delimiter:

mImport(gTable, "Text", the moviePath & "test.txt", [#FieldDelimiter:"-", #RecordDelimiter:NumToChar(14)])
This example imports a text file with "%" as field delimiter, "¬" as record delimiter, "\" as virtual carriage return and "+" as text qualifier. The character set is considered to be "MS-DOS":

mImport(gTable, "Text", the moviePath & "test.txt", [#FieldDelimiter:"%", #RecordDelimiter:"¬", #CharacterSet:"MS-DOS", #VirtualCR:"\", #TextQualifier:"+"])

Literal

stringToImport = A Lingo string or expression that resolves to a string.
optionsList = Lingo property list containing the following properties (all optional):

#FieldDelimiter	character. Default: TAB.
#RecordDelimiter	character. Default: RETURN.
#CharacterSet	"Windows-ANSI", "Mac-Standard" or "MS-DOS". Default: Matches current OS.
#VirtualCR	character: Default: NumToChar(11).
#TextQualifier	character. Default: "|"

DBF

fileToImport = Pathname of the dbf file to import.
optionsList = Optional parameter. Lingo property list containing the following properties:

#CharacterSet	"Windows-ANSI", "Mac-Standard" or "MS-DOS". Default: Matches current OS.

Example
This statement, run on Windows, imports a DBF file containing MS-DOS characters. V12 converts the MS-DOS character set to Windows character set.

Example
-- This statement imports a DBF file.
mImport(gTable, "DBF", the moviePath & "sourceData.DBF", [#CharacterSet:"MS-DOS"])

List

listToImport = Lingo list of lists. Each sub-list is equivalent to one record in your table. The first sub-list must contain your field descriptors.

Example
-- This statement imports a record from a list.
mImport(gTable, "List",[["name", "price"], ["spoon", 1.9]])

PropertyList

listToImport = Lingo property list containing the data to import. Each sub-list represents a record and each item is a property/value pair: the property is the name of the field and the value is the data contained in it.
Example
-- This statement imports a record from a propertylist.
mImport(gTable, "Propertylist",[[#name: "spoon", #price, 1.9]])

V12

fileToImport = The pathname of the V12 file to import.

	password = Password to unlock the database to import from. *tableName* = Name of the table to import. Example -- This statement imports the contents of a table named “articles”. mImport(gTable, "V12", the moviePath & "catalog.v12", "top secret", "articles"])
Access (Windows only)	*fileToImport* = The pathname of the Access file to import. *userName* = Valid username to access the MDB file, or EMPTY if the MDB file is not protected. *password* = Username's matching password, or EMPTY if the MDB file is not protected. *tablename* = Name of the table to import. Example -- This statement imports the contents of a table named “articles”. mImport(gTable, "Access", the moviePath & "catalog.mdb", "", "", "articles"])
FoxPro (Windows only)	*path* = Path to the source *.DBF file(s). Example mImport(gTable, "FoxPro", the moviePath & "dbfs")
Excel (Windows only)	*fileToImport* = The pathname of the Excel file to import. *tablename* = Name of the table to import. Example mImport(gTable, "Excel", the moviePath & "catalog.xls", "articles")
SQLserver (Windows only)	*DSN* = Valid Data Source Name. *userName* = Valid username to access the SQL Server database. *password* = Username's matching password. *tablename* = Name of the table to import. Example mImport(gTable, "SQLServer", "CatalogDSN", "Admin", "XBF48", "articles")

mNeedSelect (T)

Syntax

```
mNeedSelect(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Checks the content of *whichTable* to see if another user in "Shared ReadWrite" mode has modified it. The method will return TRUE or FALSE.

`mNeedSelect` returns TRUE if records were added, deleted or modified since the current instance last called `mSelect`. In some cases, it is a good idea to check for `mNeedSelect` on idle and refresh the displayed records when signaled to do so.

Example

```
if mNeedSelect(gTable) then
        mSelect(gTable)
        -- do whatever necessary to refresh the display
end if
```

See Also

```
mSelect (T), mSelectAll (T)
```

mOrderBy (T)

Syntax

mOrderBy(*whichTable*, *fieldName* [, *sortOrder*])

Parameters

whichTable = Instance of a V12 Table.
fieldName = Name of field to use as the sorting key.
sortOrder = "ascending" or "descending", Default = "ascending". Optional parameter.

Description

Sort the selection according to field fieldName. This method is normally called just before the `mSelect` method is used.
NOTE: when `mOrderBy` is used before `mSelectAll`, fieldName must be indexed.

Example

```
mOrderBy(gTable, "lastName") -- ascending by default.
CheckV12Error() -- this handler needs to be defined by you
mSelect(gTable)
CheckV12Error()
mOrderBy(gTable, "lastName", "ascending")
CheckV12Error()
mSelect(gTable)
CheckV12Error()

mOrderBy(gTable, "lastName", "descending")
CheckV12Error()
mSelect(gTable)
CheckV12Error()
```

See Also

```
mSelectAll (T), mSelect (T), mSetIndex (T)
```

mPackDatabase (DB)

Syntax

mPackDatabase(*whichDatabase*, *pathname*)

Parameters

whichDatabase = Instance of a V12 Database.
pathName = Pathname of the new V12 Database compact file.

Description

Reclaims the space lost because of deleted records by creating a new compact file.

Example

```
mPackDatabase (gDB, the moviePath & “PackMan.V12”)
```

mReadDBStructure (DB)

Syntax

```
mReadDBStructure(whichDatabase, inputType, inputData [, password])
```

Parameters

whichDatabase = Instance of a V12 Database.
inputType = "LITERAL", "TEXT", "V12", "DBF"
Windows only: "FoxPro", "Access", "Excel" or "SQLServer"
inputData = Pathname of template file or database descriptor expression (if *inputType* = "LITERAL")
password = Relevant only if *inputType* is "V12".

Description

Creates a new database or modifies an existing one. `mReadDBstructure` can read a definition from a string, field or variable ("LITERAL"), from a text file ("TEXT"), from another V12 database file ("V12"), from a DBF file ("DBF") or from ODBC sources on Windows.

Example

```
-- read a definition from a Director field member:
mReadDBStructure(gDB,"LITERAL",field "definition")

-- read a definition from a TEXT file:
mReadDBStructure(gDB,"TEXT", the moviePath & "definition.txt")

-- read a definition from a V12 database file:
mReadDBStructure(gDB,"V12", the moviePath & "definition.v12", "top secret")

-- read a definition from a DBF database file:
mReadDBStructure(gDB,"DBF", the moviePath & "definition.dbf")
```

See Also

```
new (DB), mBuild (DB)
```

mSelDelete (T)

Syntax

```
mSelDelete(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Deletes all the records of a selection. At the end of the operation, the selection is empty.

Example

```
-- the following will delete the current selection:
mSelDelete(gTable)
```

See Also

```
mDeleteRecord (T)
```

mSelect (T)

Syntax

```
mSelect(whichTable[, from [, #recs]])
```

Parameters

whichTable = Instance of a V12 Table.
from = The first record to be selected. Optional parameter.
#recs = The number of records to select. Optional parameter.

Description

Triggers the selection process. This is required after calls to `mSetIndex`, `mSetCriteria` and/or `mOrderBy`. If no record satisfies the search criteria, `mSelect` returns an empty selection and sets the current record to an undefined value. If the *from* and *#recs* parameters are omitted, the whole selection is obtained.

Example

```
-- select all records of the table and sort them by order of catalog number:
mSetIndex(gTable, "CatNumberNdx")
CheckV12Error() -- this handler needs to be defined by you
mSelect(gTable)
CheckV12Error()

-- select all items that cost at least $20,
-- and at most $40:
mSetCriteria(gTable, "price", ">=", 20)
CheckV12Error()
mSetCriteria(gTable, "and", "price", "<=", 40)
CheckV12Error()
mSelect(gTable)
CheckV12Error()

 -- select all items that cost at most $40
 -- and sort them by alphabetic order:
mSetCriteria(gTable, "price", "<=", 40)
CheckV12Error()
mOrderBy(gTable, "name")
CheckV12Error()
mSelect(gTable)
CheckV12Error()
```

See Also

mSetIndex (T), mSetCriteria (T), mOrderBy (T)

mSelectAll (T)

Syntax

```
mSelectAll(whichTable)
```

Parameters

whichTable = Instance of a V12 Table.

Description

Selects all the records of a table. The sorting order for the selection is the same as the most recently chosen index unless it is preceded by `mOrderby`. That index is either explicitly chosen by you (`mSetIndex`) or automatically chosen by `mSetCriteria` and/or `mOrderBy`.

Example

```
mOrderby(gTable,"price")
CheckV12Error() -- this handler needs to be defined by you
mSelectAll(gTable)
CheckV12Error()
```

See Also

```
mOrderBy (T)
```

mSelectCount (T)

Syntax

`mSelectCount(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Returns the number of records in the selection. If the selection is empty, this method returns 0.

Example

```
put mSelectCount(gTable) into field "TotalHits"
```

See Also

```
mGetPosition (T)
```

mSetCriteria (T)

Syntax

`mSetCriteria(`*`whichTable [,boolOp], fieldName`*`, `*`operator`*`, `*`value`*`)`

Parameters

whichTable = Instance of a V12 Table.
boolOp = "and" or "or".
fieldName = field which value is searched in *whichTable*.
Operator = "=", "<>", "<", ">", "<=", ">=", "starts", "contains", "wordEquals", "wordStarts".
value = the value to look for.

Description

Specifies a search criteria. A call or sequence of calls to `mSetCriteria` must be followed by a call to `mSelect` to trigger the search process. If more than one criterion is used, subsequent criteria must use the Boolean operator “and” or “or”.

Example

```
-- finds all records where the field "muffin"
-- contains "chocolate". Note that the field muffin is full-indexed.
mSetCriteria(gTable, "muffin", "wordEquals", "chocolate")

-- This instruction combines a full text search
-- in two fields with an ordinary search
mSetCriteria(gTable, "muffin", "wordEquals", "chocolate")
mSetCriteria(gTable, "or", "donut","contains", "chocolate")
mSetCriteria(gTable, "and", "name", "starts", "Shlomo")
mOrderBy(gTable, "price")--selection doesn't apply to full Index
mSelect(gTable)
```

See Also

mSelect (T), mOrderBy (T), mFind (T)

mSetField (T)

Syntax

```
mSetField(whichTable, fieldName, value)
```

Parameters

whichTable = Instance of a V12 Table.
fieldName = Name of the field who's content is modified in the current record in *whichTable*.
value = Value to assign to the field fieldName of the current record in *whichTable*.

Description

Sets the content of field *fieldName*, of the current record, to value. If value is not of the same type as *fieldName*, INM V12 Database casts it to the appropriate type. If *fieldName* is a date, the value must be a valid date in V12 Database's raw format (YYYY/MM/DD). Calls to `mSetField` must be preceded by a call to `mEditRecord` or to `mAddRecord`, and must be followed by a call to `mUpdateRecord`.

Example

```
-- editing an existing record:
mEditRecord(gTable)
CheckV12Error() -- this handler needs to be defined by you
mSetField(gTable, "description", field "myDescription")
CheckV12Error()
mSetField(gTable, "height", integer(field "height"))
CheckV12Error()
mUpdateRecord(gTable)
CheckV12Error()

-- adding a new record to the table gTable:
mAddRecord(gTable)
CheckV12Error()
mSetField(gTable, "name", "hot dog")
CheckV12Error()
mSetField(gTable, "length", 2)
CheckV12Error()
mSetField(gTable, "price", 1.95)
CheckV12Error()
mUpdateRecord(gTable)
CheckV12Error()
```

See Also

mGetMedia (T), mSetMedia (T), mGetMedia (T), mEditRecord (T), mUpdateRecord (T)

mSetIndex (T)

Syntax

```
mSetIndex(whichTable, indexName)
```

Parameters

whichTable = Instance of a V12 Table.
indexName = Name of the index to set as current index.

Description

Sets the index *indexName* as the current index.
A call to `mSetIndex` must be followed by a call to `mSelect`.
It is useless to call `mSetIndex` before setting search criteria as `mSetCriteria` selects the most appropriate index for a given query.

Example

```
-- select all records of the table and sort them by order of price:
mSetIndex(gTable, "priceNdx")
mSelect(gTable)
```

See Also

```
mSelectAll (T), mOrderBy (T)
```

mSetMedia (T)

Syntax

```
mSetMedia(whichTable, fieldName, memberRef)
```

Parameters

whichTable = Instance of a V12 Table.
fieldName = Name of the field in which media is to be stored in *whichTable*.
memberRef = Director cast member from which media is retrieved.

Description

Replaces the contents of the field *fieldName* of the current record with the cast member *memberRef*.

Example

```
-- get the media from the cast member named "Image"
-- in cast 1 and store it in the field "photo", of the current record:
mEditRecord(gTable)
mSetMedia(gTable, "photo", member "Image")
mUpdateRecord(gTable)

-- or

mEditRecord(gTable)
mSetMedia(gTable, "photo", member "Image" of CastLib "internal")
mUpdateRecord(gTable)
```

See Also

```
mSetField (T), mGetMedia (T)
```

mSetPassword (DB)

Syntax

```
mSetPassword(whichDatabase, oldPassword, newPassword)
```

Parameters

whichDatabase = Instance of a V12 Database
oldPassword = Current password of *whichDatabase*.
newPassword = New password of *whichDatabase*.

Description

Changes the current password (oldPassword) to a new one (newPassword). Either oldPassword and/or newPassword can be empty strings.

Example

```
-- change the password "very secret" to "even more secret":
mSetPassword(gDB, "very secret", "even more secret")
-- change from an empty password to "my new password":
mSetPassword(gDB, "", "my new password")
```

mSetProperty (DB)

Syntax

```
mSetProperty(whichDatabase, propertyName, value)
```

Parameters

whichDatabase = Instance of a V12 Database.
propertyName = Custom or predefined property names.
value = Value will vary depending on property being modified. (See Chart Below)

Description

Changes an existing property, or creates a new property. Special rules apply to properties that start with "string." (see manual)

Property	**Type**	**Valid values**
ProgressIndicator	Persistent, ReadWrite	"None" "With_Cancel" "Without_Cancel" "UserDefined"
ProgressIndicator .Message	Persistent, ReadWrite	A string
VirtualCR	Persistent, ReadWrite	Character to treat as virtual CR (the character itself, *not* the ASCII code)
CharacterSet	Persistent, ReadWrite	"Windows-ANSI" "Mac-Standard" "MS-DOS"
Resources	non-persistent, ReadOnly	None
CurrentDate	non-persistent,	None

	ReadOnly	
Verbose	non-persistent, ReadOnly	"on" "off"
Months	Persistent, ReadWrite	A 12-word string, one for each month of the year. Words are separated by spaces.
ShortMonths	Persistent, ReadWrite	A 12-word string, one for each month of the year. Words are separated by spaces.
Weekdays	Persistent, ReadWrite	A 12-word string, one for each month of the year. Words are separated by spaces.
ShortWeekdays	Persistent, ReadWrite	A 12-word string, one for each month of the year. Words are separated by spaces.
ErrorLog	Persistent, ReadWrite	"on" "off"
MaxLoggedErrors	Persistent, ReadWrite	"1".."100" Default: "32"
SharedRWcount	non-persistent, ReadOnly	None
Dbversion	non-persistent, ReadOnly	None
String.*Language*	non-persistent, ReadOnly	A Search/sort table
String.*Language*.Delimiters	non-persistent, ReadOnly	A string containing individual characters to be treated as delimiters.
String.*Language*.MinWordLength	non-persistent, ReadOnly	"2".."8" Default: "4"
String.*Language*.StopWords	non-persistent, ReadOnly	A string containing individual words to be treated as Stop Words. Words are separated by spaces.
FlushToDisk	ReadWrite	"true" "false"

Example 1

```
--This line sets the “months” property to a string containing
-- the month names in English
mSetProperty (gDB, "months", "January February March April May June July
August September October November December")
```

Example 2

```
-- This line sets the size of the shortest word to be
-- considered for full-indexing to 5 characters
mSetProperty (gDB, "string.MinWordLength", String(5) )

Note: See the V12 User Manual for additional information.
```

Example 3

```
-- Turn on the FlushToDisk property for one set of write operations:
mSetProperty(gdb, "flushtodisk", "true")
CheckV12Error()

        repeat with i = 1 to 1000
                mAddRecord(gTable)
                CheckV12Error()
                mSetField(gTable, "field1", i)
                CheckV12Error()
                mUpdateRecord(gTable)
                CheckV12Error()
        end repeat

mSetProperty(gdb, "flushtodisk", "false")
CheckV12Error()
```

See Also

`mGetProperty (DB)`

mUpdateRecord (T)

Syntax

`mUpdateRecord(`*`whichTable`*`)`

Parameters

whichTable = Instance of a V12 Table.

Description

Saves modifications of the current record to the database file. A call to `mUpdateRecord` must be preceded by a call to `mEditRecord` or `mAddRecord`.

Example

```
mEditRecord(gTable)
CheckV12Error()
mSetField(gTable, "name", field "name")
CheckV12Error()
mUpdateRecord(gTable)
CheckV12Error()
```

See Also

```
mEditRecord (T), mSetField (T), mSetMedia (T)
```

new (DB)

Syntax

new(Xtra "V12DBE", *databaseName*, *openMode*, *password*)

Parameters

databaseName = Name of the database file to create or open
openMode = The mode in which the database will be opened. Valid Values are "Create", "ReadWrite", "ReadOnly", "Shared ReadWrite" and "Shared ReadOnly". If *openMode* is "create", the third parameter is a password and it is stored in the database file for later reference. If *openMode* is "ReadOnly", "ReadWrite", "Shared ReadWrite" or "Shared ReadOnly", the third parameter is a password and it is checked against the one provided with "Create".
password = The password of your database. If no password is required, pass an empty string.

Description

Creates a database Xtra instance and returns a reference to it. Usually, that reference is assigned to a global variable and used throughout the Lingo script to refer to that database.
If *openMode* is "create" and new database file is created, table, field and index definitions must follow. That process must be terminated by a call to `mBuild`.
If *openMode* is "ReadOnly", data can be read but not written to the database.
If *openMode* is "ReadWrite", data can be read and written to the database.
If *openMode* is "Shared ReadWrite", data can be read and written to the database across a network by multiple users.

Examples

```
create a new database named "myBase"
-- and lock it with password "very secret":
set gDB = new(Xtra "V12DBE", "myBase", "Create", "very secret")

-- open an existing database file named "myBase" in read-only mode:
-- (i.e. the database cannot be modified).
set gDB = new(Xtra "V12DBE", "myBase", "ReadOnly", "very secret")

-- open an existing database (FirstDB.v12) and
-- clone it in the directory of the current movie:
set gDB1 = new(Xtra "V12DBE", the moviePath & "FirstDB.v12", "ReadOnly", "top
secret")
```

new (T)

Syntax

new(Xtra "V12Table", mGetRef(*whichDatabase*), *tableName*)

Parameters

whichDatabase = Reference to the V12 Database that contains *tableName*.
tableName = Name of table to open.

Description

Creates a table Xtra instance and opens the table *tableName*. New returns a reference to that Xtra instance, which is normally assigned to a global variable for later reference.

Example

```
set gDB = new(Xtra "V12DBE", the moviePath & "myBase", "ReadOnly", "very secret")
set gTable = new(Xtra "V12Table", mGetRef(gDB), "MegaTable")
```

See Also

`new (DB)`

mClose (DB)

Syntax

`whichDatabase``.mClose()`

Parameters

whichDatabase = Reference to the V12 Database you are closing

Description

Closes a V12 database Xtra instance. After calling mClose(), the global database variable that refers to the database should be set to zero.

Before closing the database instance, all related table instances should be closed.

Examples

```
-- Close all table instances first
gTable1.mClose()
gTable1 = 0
gTable2.mClose()
gTable2 = 0

-- now close the database and set its reference variable to zero
gDB.mClose()
gBD = 0
```

mClose (T)

Syntax

`whichTable``.mClose()`

Parameters

whichTable = Reference to the V12 table instance you are closing

Description

Closes a V12 table instance instance. After calling mClose(), the global variable that refers to the table should be set to zero.

Before closing a V12 database instance, all related table instances should be closed.

Examples

```
-- Close all table instances first
gTable1.mClose()
gTable1 = 0
gTable2.mClose()
gTable2 = 0

-- now close the database and set its reference variable to zero
gDB.mClose()
gBD = 0
```

V12Download (G)

Syntax

`V12Download(`*`url`*`, `*`localFile`*`, `*`password`*`, `*`completionHandler`*`, `*`statusHandler`*`, `*`reference`*`)`

Parameters

url = URL of file to download.
E.g.: "*http://www.IntegrationNewMedia.com/test.v12*".

localFile = Full pathname of the destination file. If the movie is playing in Director's authoring mode, as a projector, or as a local shockwave movie (specified by a local file name), the V12 database is downloaded to the specified location. If the movie is running as a remote shockwave movie (a shockwave movie specified by its URL), only the file name is used. The full path, if specified, is ignored and the local file is created in the Temp folder. In that case, the downloaded database is erased when the movie is closed. If the local file exists and it is not a V12 database or if its password is not supplied, `V12Download` fails and signals an errorcode.
The local file must be closed when `V12Download` is invoked.

password = Password of the database to download. If *localFile* exists, password must also be able to unlock it.

completionHandler = Lingo handler to call when download is complete. If *reference* is zero, *completionHandler* accepts a single parameter: `V12Download`'s error code. If *reference* is an actual reference to a parent script or behavior, that reference is also returned to *completionHandler*.

statusHandler = Lingo handler to call repeatedly as the download progresses. If *reference* is zero, *statusHandler* accepts a single parameter: the number of bytes downloaded so far. If *reference* is an actual reference to a parent script or behavior, that reference is also returned to *completionHandler*.

reference = reference of parent script or behavior, or zero if none is used.

Description

Downloads the remote V12 database specified by its *url* to the local hard drive. If a local V12 database of the same name already exists it is replaced by the downloaded file. The Xtra automatically ensures that it is a valid V12 database and its password is supplied.

`v12download` resumes and returns control to Director immediately after initiating the download query. It calls *statusHandler* as frequently as possible during download (generally used to display the download status in the user interface) and ends-up calling *completionHandler* when the download is completed (generally used to open the database and start working with it).

Note: This method requires that the Director Networking Xtras be installed. In Director, check the following: MODIFY->MOVIE->XTRAS... Add Network.

Example

```
on doDownloadDB
        V12Download("http://www.tagada.com/myDB.V12", the moviePath & "myDB.V12",
"very secret password", "whenDownloadComplete", "asDownloadProgresses", 0)
    -- will periodically call asDownloadProgresses
    -- will resume execution in whenDownloadComplete
V12CheckError() -- if error occurred (wrong password, non-V12 local file...),
download is aborted
end doDownloadDB


on asDownloadProgresses b
    global gDBsize -- already initialized with actual size of file to download
    if (gDBsize>0) then put (100*b/gDBsize) & "% complete" into field "Feedback"
end asDownloadProgresses

on whenDownloadComplete err
    if (err) then
        alert "Download error " & V12Error(err)
        exit
    end if
    -- start normal use of local V12 database
    set gDB = new(Xtra "V12DBE", the moviePath&"myDB.V12", "ReadOnly", "passw")
    V12CheckError()
    -- etc.
end whenDownloadComplete
```

See Also

V12DownloadInfo (G)

V12DownloadInfo (G)

Syntax

V12DownloadInfo(*url*, *infoType*, *completionHandler*, *reference)*

Parameters

url = URL of file to get information on.
E.g.: “http://www.IntegrationNewMedia.com/test.v12”.

infoType = Type of information required. Currently, only file size is supported, #size. This parameter is therefore necessary.

completionHandler = Lingo handler to call when the required information is received on the client side. If *reference* is zero, *completionHandler* accepts two parameters: `V12DownloadInfo`'s error code and the required data (in this case, file size). If *reference* is an actual reference to a parent script or behavior, that reference is also returned to *completionHandler*.

reference = Reference to parent script or behavior, or zero if none is used.

Description

Gets information about *url* before actually downloading it. At the present time, only *url*'s file size can be returned, although other attributes (such as creation date, modification date, etc.) may be supported in a future release. This feature largely depends upon the HTTP server's ability to provide the required information.

`V12downloadInfo` resumes and returns control to Director immediately after initiating the network query. It eventually calls *completionHandler* when the required information is received.

Note: This method requires that the Director Networking Xtras be installed. Go to Director and check the following: MODIFY->MOVIE->XTRAS... Add Network.

Example

```
on getRemoteDBsize
    V12DownloadInfo("http://www.tagada.com/myDB.V12", #size, "whenInfoReceived", 0)
    -- will resume execution in whenInfoReceived
end GetRemoteDBsize

on whenInfoReceived err, s -- called back from getRemoteDBsize
    global gDBsize
    if (err) then
        alert "Network error " & V12Error(err) & ". Please make sure that" & "your
network connection is up and that the requested URL exists."
        exit
    end if
    gDBsize = s
    -- can download file now with V12Dwnload
end whenInfoReceived
```

See Also

```
V12Download (G)
```

V12Error (G)

Syntax

```
V12Error([errorNumber])
```

Parameters

errorNumber = Error number of the error to retrieve. Optional parameter.

Description

If you call `V12Error` without the *errorNumber* parameter and right after calling a V12 method, it returns an accurate and contextual description of the result. When called with the *errorNumber* parameter, a generic explanation of that error code is provided. `V12Error` is global method. It is an alternate syntax to `mError`.

Example

```
set errMsg = V12Error()
set errMsg = V12Error(-30000)
```

See Also

```
V12Status (G)
```

V12Status (G)

Syntax

```
 V12Status()
```

Parameters

none

Description

Returns the error code of the last INM V12 Database method called. An error code of 0 means no error occurred. A positive code signals a warning. A negative code signals an error. Call V12Error to get a complete explanation of the problem(s) that occurred in the last method.

Example

```
mSetCriteria(gTable,"name","=","buzzlightyear")
if V12Status() then Alert V12Error()
```

See Also

```
V12Error (G)
```

mXtraVersion (S)

Syntax

```
mXtraVersion(xtra "V12DBE")
```

Parameters

None

Description

Returns the version of the INM V12 Database xtra in Director's Xtra folder. This method only works for INM V12 Database and it must be invoked as in the example below.

Example

```
put mXtraVersion(xtra "V12DBE")
-- "V12,3.4,Multi-User"
```

INM V12 Database Error Codes

-1: Selection empty

-2: Not initialized properly

-3: Internal error

-4: Bad global area

-5: Disk read error

-6: Disk Write Error

-7: Header Read Error

-8: Header Write Error

-9: The file does not exist or is already open

-10: Not closed properly

-11: No Space

-12: File already exists

-13: Not created properly

-14: Incomplete Data

-15: Bad Header

-16: Bad Node

-17: Bad Split Entry

-18: File Not Open

-19: File Not Closed

-20: No Root Node

-21: No Current

-22: Bad Index Number

-23: Bad data length

-24: Bad reference type

-25: Bad field reference

-26: Bad field pointer

-27: Bad field handle

-28: Bad field type

-29: Bad Sequence type

-30: Bad key length

-31: Bad key type

-32: Bad Duplicate type

-33: Buffer overflow

-34: Bad file specification

-35: Bad minimum extend

-36: Over demo limit

-37: File seek

-38: Log record number not used

-39: Double lock current info

-40: Double unlock current info

-41: Entry has bad data length

-42: Bad segment number

-44: Memory allocation error

-45: Data checksum error

-46: Data definition checksum error

-47: Unable to open database. The maximum of users as been reached

-48: Bad build key

-49: Duplicate key

-50: Invalid number of buffers

-51: Key too big

-52: Too many segments

-53: Bad lock current info

-81: Bad load shared library

-82: Function not loaded

-83: Function not found

-101: File locked

-102: File mode error

-103: Not enough memory or not multiuser OS

-104: Not locked

-105: Current record locked by other user

-106: Locked by self

-107: Reset error

-108: Clear schema error

-109: Bad clear byte

-110: Bad set byte

-111: Current Record already locked

-201: Bad select position number

-202: Bad field number

-203: Bad select type

-204: Bad select Op

-205: User abort

-206: Bad key number

-207: Different select types

-520: Invalid open mode

-530: Invalid parameter

-540: Bad edit mode

-550: Unknown error

-560: Invalid identifier. Valid identifiers must have at least one character

-570: Invalid identifier. First character must be alphabetic

-580: Invalid character(s) in identifier

-590: Invalid identifier length. Valid identifiers have at most 32 characters

-600: Table '%s' does not exist

-610: Field '%s' does not exist in table '%s'

-620: Field '%s' of type '%s' of table '%s' is of a type that cannot be full-indexed

-630: Invalid field type

-640: Invalid parameter. The parameter must be a valid V12base component

-650: Invalid parameter. The parameter must be a valid V12table component

-660: The database used by the table is not open

-1010: Bad table instance. Check current instance

-1030: Too many records

-1050: Invalid object

-1060: Invalid database structure

-1070: Memory allocation error

-1080: Field does not exist

-1090: Unable to read structure from database

-1100: Structure not initialized properly

-1110: Corrupted DBF file

-1120: Field does not exist. Please contact tech support

-1130: Cannot modify table

-1140: Invalid database structure. Please contact tech support

-1150: Invalid identifier

-1160: Cannot create text file. Maybe the file already exists

-1170: Cannot create DBF file. Maybe the file already exists

-1180: Cannot create DBT file. Maybe the file already exists

-1250: Field does not exist

-1260: Invalid field data. Please contact tech support

-1270: Invalid field type

-1280: Invalid field size in table

-1290: No table defined

-1330: Unable to set password

-1370: Duplicate key

-1380: Unable to create or modify a database on a locked file/volume

-1400: Database not initialized properly. Please contact tech support

-1410: Unable to pack database. File name already exists

-1420: Table already in use. Set all instances of a table to zero

-1430: MOA error in V12-DBE. Please contact Tech Support

-1440: Cannot set a void value

-1460: Cannot bind multiple fields with the same member name

-1480: Item not found in table

-1500: The current record is locked by another user

-1530: Cannot open database. This database structure is not supported by the current version of V12

-1540: One or more users opened this V12 database in Shared Read/Write mode. All Read-Only operations are suspended until the Read/Write client(s) close the database

-1810: Low-level engine not initialized

-1820: Wrong number of parameters

-1830: Invalid file name

-1840: Invalid open mode. Please consult the manual to get a description of the different open mode

-1841: '%s' is an invalid open mode. Please consult the manual to get a description of the different open mode

-1850: V12-DBE instance was not opened properly

-1860: Corrupted variables. Reboot the computer

-1870: Invalid pathname

-1880: File already exists

-1890: Cannot create database file. File already exists and is open

-1900: Error while writing header files

-1910: File does not exist or is already open

-1920: Not enough disk space

-1930: Wrong password

-1940: Cannot get password. Please contact tech support

-1960: Invalid password. Check if the password is not VOID

-1980: Cannot delete database file. Make sure the database file is not open

-2210: Invalid object

-2410: Unable to edit database structure. Database must be opened in ReadWrite mode

-2810: Unable to update database structure. Database must be opened in ReadWrite mode

-2820: Not in database structure edition mode. Call mEditDBstructure before modifying database structure

-3010: Wrong number of parameters

-3020: Invalid pathname

-3030: Empty pathname

-3210: Wrong number of parameters

-3220: Invalid descriptor type. Please consult the manual to get a description of the different descriptor type

-3230: Invalid database descriptor. Check descriptor's syntax

-3240: Unable to locate/decode password

-3250: Unable to read database structure

-3260: Field '%s' of table '%s' has an invalid index order. Valid orders are Ascending and Descending

-3270: Missing '(*' or '*)'

-3280: Unable to open TEXT file. Make sure the file is in the specified path and not used by another application

-3290: Unable to open V12 file. Make sure the file is in the specified path and not used by another application

-3300: Unable to open DBF file. Make sure the file is in the specified path and not used by another application

-3310: Unable to modify database structure. Call mEditDBstructure first

-3320: Empty file name

-3330: Missing [END] tag in database descriptor

-3340: Missing field name in table '%s'

-3350: Field '%s' of type Media cannot be indexed

-3360: Missing [TABLE] tag in database descriptor

-3370: Missing table name in database descriptor

-3380: Invalid field name in table '%s'

-3390: Field '%s' already exists in table '%s'

-3400: Field '%s' of type '%s' of table '%s' is of a type that cannot be full-indexed

-3410: Invalid field type in table '%s'

-3420: Maximum number of indexes reached. The maximum is %ld

-3430: Index '%s' already exists in table '%s'

-3440: Invalid index type in index '%s' of table '%s'

-3450: Missing field name for index '%s' in table '%s'

-3460: Field '%s' set in index '%s' of table '%s' does not exist

-3470: Missing order for index '%s' of table '%s'. Valid orders are Ascending and Descending

-3480: Invalid field name. '%s' is a reserved word

-3490: Field '%s' of table '%s' has an invalid field type

-3500: Invalid descriptor type. Please consult the manual to get a description of the different descriptor type

-3510: Empty database descriptor

-3520: Table '%s' already exists

-3530: Field '%s' does not exist in table '%s'

-3540: Unable to edit database structure. Database must be opened in Create or ReadWrite mode

-3550: Not in database structure edition mode. Call mEditDBstructure first

-3570: Invalid DBF file format

-3580: Buffer size must be omitted for fields of type '%s'

-3590: Invalid field size in table '%s'

-3600: First character of table '%s' must be alphabetic

-3610: Unable to modify database structure. Database must be opened in Create or ReadWrite mode

-3630: Invalid table identifier. Verify database descriptor

-3640: %s

-3810: Wrong number of parameters

-3820: Invalid table name

-3830: Table '%s' already exists

-3840: Unable to create new table. Call mEditDBstructure before creating new tables

-3850: Empty table name

-3860: Unable to edit database structure. Open database in Create or ReadWrite mode

-3870: Table '%s' contains invalid characters

-3880: First character of table '%s' must be alphabetic

-3890: Table '%s' has an invalid identifier length. Valid identifiers have at most 32 characters

-3900: Cannot create table '%s'. The maximum number of table(s) is '%ld'

-4010: Wrong number of parameters

-4020: Invalid table name

-4030: Invalid field name

-4040: Invalid field type

-4050: Invalid buffer size

-4060: Table '%s' does not exist

-4070: Database structure not created properly

-4080: Field '%s' already exists in table '%s'

-4090: Unable to create new field. Call mEditDBstructure and mCreateTable before creating new fields

-4100: Empty table name

-4110: Empty field name

-4120: Invalid buffer size. Buffer size must be greater than zero

-4140: Buffer size not required for fields of type '%s'

-4160: Unable to use '%s' as a field name. This is a reserved word

-4170: Invalid index type. Valid index types are indexed and full-indexed

-4180: Unable to edit structure of table '%s'. Table already built

-4190: Unable to edit database structure. Open database in Create or ReadWrite mode

-4200: Field '%s' of table '%s' contains invalid characters

-4210: First character of field '%s' in table '%s' must be alphabetic

-4220: Field '%s' of table '%s' has an invalid identifier length. Valid identifiers have at most 32 characters

-4230: Field '%s' of table '%s' cannot be indexed. It must have at most 29 characters to be indexed

-4240: Cannot create field '%s'. The maximum number of field(s) is '%ld'

-4510: Wrong number of parameters

-4520: Table '%s' does not exist

-4530: Field '%s' does not exist in table '%s'

-4540: Database structure not created properly

-4550: Invalid table name

-4560: Invalid field name

-4570: Invalid index name

-4580: Invalid index type. Valid types are Duplicate and Unique

-4590: Invalid index order. Valid orders are Ascending and Descending

-4591: '%s' is an invalid index order. Valid orders are Ascending and Descending

-4600: Unable to create index. Call mEditDBstructure, then create new tables and new fields before creating new indexes

-4610: Maximum number of indexes reached. The maximum is %ld

-4620: Empty table name

-4630: Empty index name

-4640: Empty field name

-4650: Empty index type. Valid types are Unique and Duplicate

-4660: Empty index order. Valid orders are Ascending and Descending

-4670: Field '%s' already used in index '%s' of table '%s'

-4680: Unable to edit database structure. Database must be opened in Create or ReadWrite mode

-4690: Field '%s' of type Media specified in table '%s' cannot be indexed

-4700: Unable to edit structure of table '%s'. Table already built

-4710: Cannot create compound index '%s'. Limited to '%ld' field(s) per index

-4910: Unable to delete table. Database must be opened in Create or ReadWrite mode

-4920: Unable to edit database structure. Call mEditDBstructure first

-4930: Table '%s' does not exist

-4940: Unable to open table

-4950: Empty table name

-4960: Table already in use. Set all instances of a table to zero before deleting it

-4970: Cannot delete table. Database must be opened in Create or ReadWrite mode

-5110: Database structure not created properly

-5120: Missing index. Table '%s' must contain at least one index

-5130: Unable to edit database structure

-5140: Unable to modify database structure. Use mEditDBStructure and mUpdateDBStructure to change a database structure

-5150: Unable to build database. Database must be opened in Create or ReadWrite mode

-5160: Unable to update database structure. At least one index per table is required

-5170: Unable to build database structure. At least one index per table is required

-5180: Invalid build option. Please consult the manual to get a description of the

-5410: Wrong number of parameters

-5420: Invalid password. Password should not exceed 32 characters

-5430: Invalid character(s) in password

-5440: Wrong password. '%s' does not match with the current password

-5450: Unable to write to database. Database must be opened in ReadWrite mode

-5610: Wrong number of parameters

-5620: Database structure not created properly

-5630: Invalid output format. Please consult the manual to get a description of the different output formats

-5640: Table '%s' does not exist

-5650: Field '%s' does not exist in table '%s'

-5660: Can only get size information on fields of type Media or String

-5670: Invalid table name

-5680: Invalid field name

-5690: Empty table name

-5700: Empty field name

-5810: Property does not exist

-5820: Invalid property value. Please consult the manual to get a description of the property values

-5830: Missing apostrophe. A sub-string was left open-ended

-5840: Unable to write to database. Database must be opened in ReadWrite mode

-5850: Cannot delete or set the property value to blank

-5860: Cannot modify the property value

-5870: Cannot change verbose value. Invalid verbose type

-5880: Cannot set weekdays. Invalid number of days.

-5890: Cannot set months. Invalid number of months

-5900: Cannot modify property. The string type associated does not exist

-5910: Cannot modify property. The string type associated is already used

-5920: Cannot modify MinWordLength property. This property must be greater than 0 and smaller than 100

-5930: Cannot define new string type. String type names cannot contains periods (dots)

-5940: Cannot set property name. Too many characters

-5950: Cannot set new string type. The maximum number of custom string types is reached

-5960: Invalid log value. Must be set to 'on' or 'off'

-5970: Invalid maximum value. Must be a number between 1 and 1000

-5980: Invalid progress indicator value

-5990: Invalid Character Set value. Must be: Ms-Dos, Mac-Standard, Windows-ANSI

-6000: Invalid time stamp value

-6010: Property does not exist

-6110: Memory allocation error

-6120: Wrong number of parameters

-6130: V12base instance not properly opened

-6140: Invalid table name

-6150: Missing table definition in database. At least one table must be defined in a V12 database

-6160: No such table in database

-6170: Failed to open table. Database file not open.

-6180: Table was defined but not written to database

-6190: Table not found

-6200: Invalid object

-6210: Empty table name

-6220: Cannot create table instance. Only one table instance can be created.

-6230: The V12 database you are trying to access has been closed.

-6240: The V12 table you are trying to access has been closed."

-6410: Wrong number of parameters

-6420: Invalid export type

-6430: Invalid pathname

-6440: Invalid field delimiter

-6450: Invalid record delimiter

-6460: Invalid field name

-6470: Field '%s' does not exist

-6480: Invalid record delimiter length

-6490: Invalid field delimiter length

-6810: Wrong number of parameters

-6820: Wrong number of parameters

-6830: Wrong number of parameters

-6840: Parameter #2 should be either a valid pathname or a valid source type

-6850: Invalid pathname

-6860: Invalid record delimiter

-6870: Invalid field delimiter

-6880: Invalid import type. Please consult the manual to get a description of the different import type

-6890: Import error

-6900: Unable to open DBF file. Check pathname

-6910: Unable to open TEXT file. Check pathname

-6920: Empty file name

-6931: Field %ld does not exist

-6940: Empty table name

-6950: Database not properly initialized

-6970: Invalid DBF file format

-6980: Field and record delimiters must be different

-6990: Unable to import data. Database must be opened in ReadWrite mode

-7050: Cannot use line feed (LF) as field delimiter

-7070: Duplicate key occurred at line '%ld'

-7090: Cannot delete record of line '%ld'

-7100: %s

-7110: Data of field '%s', record '%ld', is longer than the '%ld' allocated for it and was truncated

-8410: Index '%s' does not exist

-8420: Invalid index name

-8430: Empty index name

-8610: Wrong number of parameters

-8620: Invalid field name

-8630: Invalid operator

-8640: Invalid field type

-8650: Invalid operator

-8660: Invalid operator

-8670: Field does not exist

-8680: No memory available

-8690: Empty field name

-8700: Operator not allowed for this type of field

-8710: Field '%s' is not full-indexed

-8720: Field '%s' does not exist

-8730: Word length is smaller than the minimum of indexed words (%ld)

-8740: Cannot specify boolean operator in first criteria

-8750: String must have at least one character

-8760: Maximum number of criteria reached

-9010: Wrong number of parameters

-9020: Field '%s' does not exist

-9030: Invalid index order

-9050: Empty index name

-9410: Wrong number of parameters

-9430: Outside of selection range

-9440: Invalid parameter(s). Please consult the manual to get a description of the different parameters

-9450: Cannot limit selection. Cannot used WordEquals or WordStarts twice

-9610: Unable to delete selection. Database must be opened in ReadWrite mode

-9620: Unable to delete record #%ld. This record is locked by another user

-9810: Invalid result

-9820: Selection empty

-10010: Field does not exist

-10020: Invalid field name

-10030: Empty field name

-10040: Field '%s' does not exist

-10060: Wrong number of parameters

-10410: Invalid value. Please contact tech support

-10440: Parameter incompatible with type of destination field

-10450: Invalid field name

-10460: Unable to modify data. Call mEditRecord first

-10470: Media exceeds buffer size declared upon database creation

-10480: Empty field name

-10490: '%s' is an invalid date

-10500: Field '%s' does not exist

-10510: Media of field '%s' cannot be saved. The media is too big.

-10810: Wrong number of parameters

-10820: Empty media field

-10830: Empty field name

-10840: Field '%s' does not exist

-10850: CastLib %ld does not exist

-10860: Unable to find member '%s'

-10870: The parameter must be a (cast) member

-10890: Cannot download media file

-11010: Wrong number of parameters

-11020: Cast member empty

-11030: Invalid field name

-11040: Empty field name

-11050: Empty media field

-11060: Field '%s' does not exist

-11070: Unable to modify data. Call mEditRecord first

-11080: Unable to import this kind of media

-11710: Unable to edit record. Database must be opened in ReadWrite mode

-11910: Cannot write data. Call mEditRecord first

-11920: Duplicate key '%s'. Check the following field(s): %s

-12130: Unable to delete record. Database must be opened in ReadWrite mode

-12710: Wrong number of parameters

-12730: Cast '%s' does not exist

-12731: Cast %ld does not exist

-12740: Cast member %s does not exist

-12741: Cast member %ld does not exist

-12750: Invalid member identifier

-13010: Wrong number of parameters

-13025: Field '%s' is not bound. Call mBindField first

-13030: Cast '%s' does not exist

-13035: Cast %ld does not exist

-13045: Cast member %s %ld does not exist

-13050: Field '%s' does not exist

-13410: Wrong number of parameters

-13420: Invalid member identifier

-13610: Wrong number of parameters

-13630: Unable to find Field '%s'

-13640: Cast '%s' does not exist

-13650: Cast %ld does not exist

-13660: Cast member '%s' does not exist

-13670: Cast member %ld does not exist

-13680: Unable to update record. Database must be opened in ReadWrite mode

-14010: Wrong number of parameters

-14020: Field '%s' does not exist

-14030: Cast '%s' does not exist

-14040: Cast member %s does not exist

-14041: Cast member %ld does not exist

-14050: Wrong number of parameters. Please consult the manual

-14060: You must specify a CastLib

-14070: Cast member '%s' not found

-14080: Invalid CastLib identifier

-14410: Wrong number of parameters

-14420: Field '%s' does not exist

-14430: Invalid parameter. Third parameter cannot be of type String

-14440: Invalid member identifier

-14450: Invalid parameter 3, attempt to get the binding type

-14460: Cannot bind field. The member '%ld' is already bound

-14810: Wrong number of parameters

-14850: Database structure not created properly

-14860: Invalid table name

-14870: Invalid field name

-14890: Field '%s' does not exist in table '%s'

-14910: Unable to create full-index. Call mEditDBstructure first

-14920: Empty table name

-14930: Empty field name

-14940: Field '%s' of type '%s' specified in table '%s' cannot be full-indexed

-14950: Unable to edit database structure. Database must be opened in Create or ReadWrite mode

-14960: Unable to edit structure of table '%s'. Table already built

-15210: Conflicting Add/Edit mode. Call mUpdateRecord before creating a new record

-15230: Unable to add record. Database must be opened in ReadWrite mode

-15610: Wrong number of parameters

-15620: Only fields type Integer, Float, and Date can be formatted

-15630: Invalid data format

-15650: Missing apostrophe. A sub-string was left open-ended

-15660: Too may periods (.) in format specifier. At most one period is allowed

-15670: Empty field name

-15680: Cannot set field format. 200 is the maximum format length

-15810: Wrong number of parameters

-15820: Invalid output format. Please consult the manual to get a description of the different output formats

-15830: Error number required

-16010: Wrong number of parameters

-16020: Invalid pathname

-16030: Empty pathname

-16040: Invalid new pathname

-16050: Empty new pathname

-16410: Wrong number of parameters

-16420: Invalid table name

-16430: Empty table name

-16440: Invalid old field name

-16450: Empty old field name

-16460: Cannot rename field. Database must be opened in Create or ReadWrite mode

-16470: Table '%s' does not exist

-16480: Field '%s' does not exist in table '%s'

-16490: Unable to rename field '%s' of table '%s'. Table already built

-16500: Unable to modify database structure. Call mEditDBstructure first

-16510: Cannot rename field. Field '%s' already exist in table '%s'

-16810: Field '%s' does not exist

-16820: Invalid output format. Please consult the manual to get a description of the different output formats

-16830: Invalid start position. Must be a number greater than zero (0) and smaller than the number of records in the selection

-16850: Invalid field delimiter

-16860: Invalid record delimiter

-16870: Invalid field name

-17210: Invalid merge type

-17220: Invalid merge option

-17230: Cannot merge data. Data transfer not complete yet

-17240: Cannot merge data. Data was not properly download

-17410: Invalid output format. Check manual for possible output formats

-17420: Field '%s' does not exist

-17430: Field '%s' of type Media cannot be retrieved with mGetUnique

-17440: Invalid field name

-17810: Wrong number of parameters

-17820: No record found

-17860: Invalid search option. Check manual for possible search options

-18010: Wrong number of parameters

-18020: Cannot download V12 file: Download not supported

-18030: Cannot download V12 file. Cannot initialize network services

-18040: Cannot download V12 file. Check URL or network connection

-18050: The downloaded file was not a valid V12 database, or you did not provide the right password. It was deleted

-18060: Cannot overwrite local file. Wrong password or invalid V12 database

Warnings

1390: File is open in ReadOnly mode. Check if file/volume is read-only

1450: Cannot update bound fields. Not in ReadWrite mode

1470: No field match. Data was not copied

1510: File is open in ReadOnly mode. Cannot open in Shared ReadOnly mode

1520: File is open in ReadWrite mode. Cannot open in Shared ReadWrite mode

1550: File is open in Shared ReadOnly mode. Check if file/volume is read-only

1950: Old database format. Opened it in ReadOnly mode

1970: This database is still in demo mode. To legalize it, please open it once in ReadWrite mode

2420: Already in database structure edition mode. Call mUpdateDBstructure before calling mEditDBstructure again

3620: Unsupported dbf field type

4130: Fields of type media cannot be indexed

4150: Fields of type '%s' cannot be full-indexed

6500: Unable to export binary field type. Field '%s' not exported in DBF file

6510: Field '%s' has been truncated to ten characters

6520: Selection empty. No current record

6530: Field '%s' of type Media cannot be export

6930: Field '%s' does not exist

6960: '%s' is an invalid date. Default date has been set

7000: Field '%s' at line '%ld' is longer than the '%ld' allocated for it and was truncated

7010: Too many field delimiters at line '%ld'. The extra data was ignored

7020: Missing field delimiter(s) at line '%ld'. Some fields were set to default data

7030: Field '%s' is defined of type media in your database. Use mSetMedia to store data in it

7040: The field '%ld' of your definition has been truncated. Exceed the maximum number of characters for a field (32)

7060: DBF field '%s' of type '%s' does not match the V12 type '%s'

7080: Unsupported dbf field type7210: Selection empty. No current record

7410: Selection empty. No current record

7610: Unable to select records beyond end of selection. %ld record(s) in selection

7620: Selection empty. No current record

7810: Unable to select records preceding first record of selection

7820: Selection empty. No current record

9420: Selection empty. No current record

Appendix 1: Database Creation and Data Importing Rules

Following are examples of how to work with existing databases from a variety of vendors, in order to use them in INM V12 Database.

As well, we have outlined standard Rules that can be referenced when you are trying to import the structures or the data from a specific database format.

The basic steps for converting a database from another database format into V12 are as follows:

8 Determine and/or Import the *structure* of the original database(s) into a readable format (using `mReadDBStructure`).

9 Create the V12 database and indexes, based on the structure of the original database(s).

10 Import the *data* from the original database(s) into the V12 databases you created in Step 2 (using mImport).

Below are examples of how to read the database structure and import files from the following text or database formats, as well as rules for each:

Text Files

mReadDBstructure from a Text File

To read a database descriptor into INM V12 Database, use the following Lingo statement:

```
            mReadDBStructure(gDB, "TEXT", File_Pathname)
```

Assuming that the name of the database descriptor's filename is "Def.txt", the following Lingo code creates a new INM V12 Database file named "Catalog.V12" and structures it as described in "Def.txt".

```
on CreateDatabase
         gDB = New(Xtra "V12dbe", the MoviePath &
           "Catalog.V12", "create", "top secret")
  CheckV12Error()
  mReadDBStructure(gDB, "TEXT", the MoviePath & "Def.txt")
  CheckV12Error()
  mBuild(gDB)
  CheckV12Error()
   gDB.mClose()
  gDB=0
end CreateDatabase
```

Importing a Text File

The imported text file must begin with a field descriptor line. A field descriptor is a sample record that contains the names of the fields in which subsequent rows of data must be formatted (see Field descriptors in Step 2: Prepare the Data). The field names used in the field descriptor line must match those supplied to the mReadStructure method. However, these fields can be listed in any order. Some of them can even be omitted.

Syntax:

```
mImport(gTable, "TEXT", FileName [, Options])
```

where `FileName` is the pathname of the text file to import, and `Options` is an optional Lingo Property list such as:

```
[ #FieldDelimiter:TAB, #RecordDelimiter:RETURN,
#CharacterSet:"Windows-ANSI", #VirtualCR:NumToChar(11),
#TextQualifier:QUOTE ]
```

`Options` may contain some or all of the properties below, or can even be empty:

- `#FieldDelimiter` determines which character is used to delimit fields in the text file. The default character is TAB (ASCII #9).
- `#RecordDelimiter` determines which character is used to delimit records in the text file. The default character is RETURN (ASCII #13). If the Text file contains Carriage Returns (ASCII #13) followed by Line Feeds (ASCII #10) as records delimiters, Line Feeds are automatically ignored.
- `#CharacterSet` is one of "Mac-Standard", "Windows-ANSI" or "MS-DOS". It determines which character set the Text file is encoded in. Usually, Text files exported on MacOS are encoded in the Mac-Standard character set, and Text files exported on Windows are encoded in the Windows-ANSI character set. See Character Sets in Step 2: Prepare the Data. The default character set is the one defined by the `CharacterSet` property (see CharacterSet in Properties of Databases).

- `#VirtualCR` determines which character is used as a Virtual Carriage Return, and thus must be converted to ASCII #13 after importing (see Virtual carriage returns in Step 2: Prepare the Data). The default character is the one defined by the `VirtualCR` property, which is usually ASCII #11 (see VirtualCR in Properties of Databases).
- `#TextQualifier` determines which character is used to begin and end each Text field. Those qualifiers delimit the field so to allow it to contain special characters, including those used as field and record delimiters. Text qualifiers are removed after importing the file. See Text qualifiers in Step 2: Prepare the Data. The default text qualifier is "|"

For example, this instruction imports the Text file "myTextData.txt" located in the same folder as the current movie into `gTable` with all the default options (field delimiter = TAB, records delimiter = RETURN, Character set = the current operating system's, virtual CR = ASCII #11, Text Qualifier = "|").

```
mImport(gTable, "TEXT", the MoviePath & "myTextData.txt")
```

This example imports the Text file "myFile.txt" where "%" is used as a field delimiter and "\" as a record delimiter.

```
mImport(gTable, "TEXT", the MoviePath & "myTextData.txt",
   [#FieldDelimiter:"%", #RecordDelimiter:"\"] )
```

Literals

mReadDBstructure from a Literal

A literal is either a Director member of type *Field* or a Lingo variable that actually contains the database descriptor (as opposed to containing the pathname of the descriptor Text file). Building a database from a literal description is very similar to the building it from a text file. The literal must contain the database descriptor as defined in Database descriptor. The Lingo script to build the database is:

```
mReadDBStructure(gDB, "LITERAL", Variable_or_Field_Name)
```

For example, assume that the Director member of type Field and named "descriptor" contains a database descriptor; this example creates an INM V12 Database file compliant to that description.

```
on CreateDatabase
   gDB = New(Xtra "V12dbe", the MoviePath&"Catalog.V12",
   "create", "top secret")
   CheckV12Error()
   mReadDBStructure(gDB, "LITERAL", field "Descriptor")
   CheckV12Error()
   mBuild(gDB)
   CheckV12Error()
  gDB.mClose()
  gDB=0
end CreateDatabase
```

Import from a Literal

Sometimes, you need to process data with Lingo before importing it into a INM V12 Database table. A convenient place to store such data is a Director member of type Field. mImport allows you to import the content of such a field through this syntax:

```
mImport(gTable, "LITERAL", DirMemberName_or_variable, [,
   Options])
```

where DirMemberName_or_variable is an expression of type Text, as in:

```
Field "myData"
the text of member "yada yada"
"Field-1,Field-2,Field-3" &RETURN& "12,14,16"&RETURN& "54,12,89"
```

and Options is a property list identical to the one used for importing Text files (see Importing a Text File above).

Following is an example of a Director field containing data to split into INM V12 Database fields and records (assume the name of the field is "Discounts"):

Level-1,Level-2,Level-3

12,14,16

45,58,72

33,56,68

224,301,451

This instruction imports the above Director field to gTable:

```
mImport(gTable, "LITERAL", field "Discounts", ",", RETURN)
```

Lingo Lists or Property Lists

Importing from a Lingo List or Property List

Lingo list, or a Lingo Property List can easily be imported to V12 tables through mImport. This is very convenient for the conversion of projects that use Lingo lists to manage data and that have become difficult to debug and maintain.

It is also convenient to import XML documents into V12 tables.

Syntax:

```
mImport(gTable, "List", theList)
mImport(gTable, "PropertyList", thePropertyList)
```

where:

- theList is a Lingo list of lists. The first element is a list containing the names of the V12 fields to which subsequent items must be imported, in the right order. If the first item of the list contains field names that are not present in the current V12 table, the corresponding data is ignored.

- `thePropertyList` is a Lingo list of property lists, where properties have the same names as the V12 fields into which the corresponding data must be imported.

Examples of valid Lingo lists:

```
[ ["LastName", "FirstName", "Age"], ["Cartman", "Eric", 8],
   ["Testaburger", "Wendy", 9], ["Einstein", "Albert", 75] ]

[ ["CatalogNumber"], [8724], [9825], [1745] ]
```

Examples of valid Property lists:

```
[ [#LastName:"Cartman", #FirstName:"Eric", #Age:8 ],
  [#FirstName:"Wendy", #LastName:"Testaburger", #Age:9 ]
  [#LastName:"Einstein", #FirstName:"Albert"] ]

[ [#CatalogNumber:8724], [#CatalogNumber:9825],
   [#CatalogNumber:1745] ]
```

V12 DBE files

mReadDBstructure from INM V12 Database

Any INM V12 Database file can be used as a template for the creation of a new INM V12 Database file, provided you know the password to unlock it. The syntax is as follows:

```
mReadDBStructure(gDB, "V12", FileName, Password)
```

This example uses the database "Catalog.V12" as a template for a new database named "Specials.V12".

```
on CreateDatabase
           gDB = New(Xtra "V12dbe", the MoviePath&"Specials.V12",
              "create", "MyNewPassword")
   CheckV12Error()
           mReadDBStructure(gDB, "V12", the
              MoviePath&"Catalog.V12", "top secret")
   CheckV12Error()
   mBuild(gDB)
   CheckV12Error()
    gDB.mClose()
   gDB=0
end CreateDatabase
```

`mReadDBStructure` reads the *structure* of a INM V12 Database file, not its content. To import the content of a INM V12 Database file, see Importing from another INM V12 Database and Add records to a database.

Importing from another INM V12 Database file

Data can be imported from one V12 table into another. The name of the source table need not necessarily match the name of the destination table. However, field names must match. Non-matching field names are ignored. If the source and destination tables have different indexes, the destination table's indexes are used.

Syntax:

```
mImport(gTable, "V12", FileName, password, TableName)
```

where `FileName` is the pathname of the V12 database to import from, `password` is the password to unlock it and `TableName` is the name of the table to import.

Example:

```
mImport(gTable, "V12", the MoviePath & "Catalog.V12", "top
   secret", "articles")
```

If two fields have the same name but are of different types when importing data from a INM V12 Database file, `mImport` tries to typecast the data fields.

DBF (Database Format)

mReadDBstructure from a DBF File

A DBF file alone represents a flat file, thus a single INM V12 Database table. A DBF file can be used as a template for an INM V12 Database table in much the same way as a text file or literal can. The name of the created INM V12 Database table is identical to the DBF filename without the ".DBF" extension. The syntax is:

```
mReadDBStructure(gDB, "DBF", File_Pathname)
```

For a DBF file to be used as a complete and valid INM V12 Database table descriptor, at least one index must be defined. If that index is defined by an IDX or NDX file located in the same folder as the DBF file, `mReadDBstructure` detects its presence and automatically defines an index for that field in the current table.

This example uses the file VIDEO.DBF as a template to build a table named "video" in the V12 Database file named "VideoStore.V12". The structure of the file VIDEO.DBF is as follows:

Field	**Type**	**Width**
TITLE	Character	30
DESCRIPT	Memo	10
RATING	Character	4
TYPE	Character	10
DATE_ARRIV	Date	8
AVAILABLE	Logical	1
TIMES_RENT	Numeric	5
NUM_SOLD	Numeric	5

Two index files named TITLE.IDX and TYPE.IDX are available in the same folder as VIDEO.DBF.

The Lingo script is as follows:

```
on CreateDatabase
   gDB = New(Xtra "V12dbe", the MoviePath&"VideoStore.V12",
   "create", "")
   CheckV12Error()
   mReadDBStructure(gDB, "DBF", the MoviePath & "Video.DBF")
   CheckV12Error()
   mBuild(gDB)
   CheckV12Error()
   put mDumpStructure(gDB)
    gDB.mClose()
   gDB=0
end CreateDatabase
```

Tip: INM V12 Database does not check the validity of the content of the index file; therefore you can fool it into creating an index for a field named "MyField" by creating an empty file named "MyField.IDX" in the same folder as your DBF file.

The resulting V12 Database file can be verified immediately with `mDumpStructure` (see View the structure of a database). The following is a sample output from `mDumpStructure`:

```
[TABLE]
Video
[FIELDS]
TITLE String 30
DESCRIPT String 30000
RATING String 4
TYPE String 10
DATE_ARRIV Date
AVAILABLE Integer
TIMES_RENT Integer
NUM_SOLD Integer
[INDEXES]
TitleNdx duplicate TITLE ascending (* Default Index *)
TypeNdx duplicate TYPE ascending
[END]
```

Importing from a DBF File

Importing a DBF file is similar to importing text files, except that you cannot specify a subset of fields to import: all the fields in the DBF file must be imported. The field names of the DBF file must match those in the destination INM V12 Database table. Non-matching field names are ignored during the importing process and a warning is reported by `V12Error` (see Errors and Defensive Programming).

Note: DBF is an antiquated file format. It is always assumed to be encoded in the MS-DOS character set. When importing DBF files, make sure to assign the right Character Set. See CharacterSet in Properties of Databases.

Syntax:

```
mImport(gTable, "DBF", FileName [, Options])
```

where `FileName` is the pathname of the DBF file to import, and `Options` is an optional Lingo Property list containing the `#CharacterSet` property:

- `#CharacterSet` is one of "Mac-Standard", "Windows-ANSI" or "MS-DOS". It determines which character set the DBF file is encoded in. Most systems automatically encode DBF file in the MS-DOS character set. See Character Sets in Step 2: Prepare the Data. The default character set the one defined by the `CharacterSet` property (see CharacterSet in Properties of Databases). It is normally "Windows-ANSI" on the Windows version of INM V12 Database and "Mac-Standard" on the Macintosh version of INM V12 Database.

Example:

```
mImport(gTable, "DBF", the MoviePath&"Pier1-Import.DBF",
   [#CharacterSet:"MS-DOS"])
```

If a field in the destination table has the same name as a field in the source DBF file, but is of a different type, `mImport` tries to typecast the data to match the destination field type. When importing data from a DBF file that contains Memo fields, the corresponding DBT files are automatically processed and imported by INM V12 Database. See Dealing with dates and *DBF (Database Format) Rules* below for more details on DBF files and data importing rules.

DBF (Database Format) Rules

The following rules apply to the translation of DBF file structures:

DBF field type	Translated to V12 field type	Notes
Character	String	Buffer size = size of field in DBF file
Integer	Integer	
Numeric with no digit after fixed point	Integer	
Numeric with one or more digits after fixed point	Float	
Float	Float	
Double	Float	
Currency	Float	On Windows 3.1 and Mac68K, acceptable values are in the range -2^k to 2^k-1, where k = 31 minus the number of decimal places.
Date	Date	
DateTime	Date	Data cannot be converted from fields of type DateTime. Only the default date (1900/01/01) is imported.
Logical	Integer	FALSE values are translated to *0*s, TRUE values to *1*s and undefined values (represented by "?" in the DBF file) to *-1*s
Media	String	Buffer size = 32K
General	Ignored	
Character-Binary	Ignored	
Memo-Binary	Ignored	

Memo fields are those typically used to store text longer than 255 characters. Memo fields can also store binary data of arbitrary formats: Binary formatted memo fields *cannot* be imported directly into INM V12 Database files. When importing standard ASCII data from a DBF file that contains Memo fields, the corresponding DBT files are automatically processed by INM V12 Database.

Microsoft FoxPro

mReadDBstructure from FoxPro (Windows Only)

A FoxPro database is a directory containing a collection of DBF files along with their index files. A directory containing one or more MS FoxPro files can be collectively used as a template for a V12 database. The FoxPro ODBC driver is required to perform this operation. The names of your FoxPro files and their field names must be valid V2-DBE identifiers (see Defining identifiers in Step 1: Decide on a Data Model).

Syntax:

```
mReadDBStructure(gDB, "FoxPro", DirectoryPath)
```

where `DirectoryPath` is the path to a directory — not a file. Thus, it must necessarily end with a "\".

Example:

```
on CreateDatabase
  gDB = New(Xtra "V12dbe", the MoviePath&"myDB.V12", "create",
  "secret")
  CheckV12Error()
  mReadDBStructure(gDB, "FoxPro", the MoviePath&"FoxDB\")
  CheckV12Error()
  mBuild(gDB)
  CheckV12Error()
   gDB.mClose()
  gDB=0
end CreateDatabase
```

Once the corresponding V12 database structure is created, with mReadDBStructure, the data from your FoxPro tables can be imported.

Importing from Microsoft FoxPro (Windows only)

Fox Pro (*.DBF) files can be imported to V12 tables provided a MS FoxPro ODBC driver is present on your PC. No DSN (Data Source Name) is required.

Syntax:

```
mImport(gTable, "FoxPro", FileName)
```

where `FileName` is the path to the source *.DBF file. Always make sure to set INM V12 Database's CharacterSet property to the encoding that matches your DBF file's (see CharacterSet in Properties of Databases).

Example:

```
          mImport(gTable, "FoxPro", the pahtname&"Results.dbf", TableName)
```

Converting a FoxPro database into a V12 database is a two-step process: First, create the V12 database, and then import data to each of its tables with `mImport`, as explained above.

FoxPro (Microsoft Fox Professional Format) Rules

The following rules apply to the translation of FoxPro databases to V12 databases:

FoxPro field type	Translated to V12 field type	Notes
Character	String	Buffer size is the size of the field in the DBF file
Integer	Float	
Numeric	Float	
Float	Float	
Double	Float	
Currency	Float	
Date	Date	
DateTime	Date	Data cannot be converted from fields of type DateTime. Only the default date (1900/01/01) is imported.
Logical	Integer	
Memo	String	Buffer size = 32K
General	String	Buffer size is the size of the field in the DBF file
Character-Binary	String	Buffer size is 32K
Memo-Binary	String	Buffer size is 32K

Microsoft Access

mReadDBstructure from MS Access (Windows Only)

MS Access databases can be used as templates to V12 databases. Like INM V12 Database, MS Access can store multiple tables per database. `mReadDBstructure` imports the structure of all such tables to INM V12 Database. The MS Access ODBC driver is required to perform this operation.

The names of the tables and fields of your MS Access file must be valid INM V12 Database identifiers (see Defining identifiers in Step 1: Decide on a Data Model).

Syntax:

```
mReadDBStructure(gDB, "Access", FileName, Username, Password)
```

where:

- `FileName` is the path to the *.MDB file,
    - `Username` is a valid user name to access the MDB file, or EMPTY if the MDB file is not protected,
    - `Password` is Username's matching password, or EMPTY if the MDB file is not protected.

Once the corresponding V12 database structure is created, the data from your MS Access tables can be imported.

Import from Microsoft Access (Windows only)

MS Access (*.MDB) files can be imported to V12 databases, one table at a time. A MS Access ODBC driver must be present but no DSN (Data Source Name) is required.

Syntax:

```
mImport(gTable, "Access", FileName, UserName, Password,
   TableName)
```

where:

- `FileName` is the path to the source *.MDB file,
- `Username` is a valid user name to access the MDB file, or EMPTY if the MDB file is not protected,
- `Password` is Username's matching password, or EMPTY if the MDB file is not protected.
- `TableName` is the name of the table to import.

Converting an MS Access database into a V12 database is a two-step process: First, create the V12 database (see mReadDBstructure from MS Access). Then, import data to each of its tables with `mImport`, as explained above.

Generally, MS Access databases are encoded in the Windows ANSI character set. Thus, you must make sure that the `CharacterSet` Property is properly set to "Windows-ANSI" before importing the data. ("Windows-ANSI" is the default setting for the `CharacterSet` property. See CharacterSet in Properties of Databases).

Microsoft Access Rules

The following rules apply to the translation of MS Access file structures to V12 databases:

MS Access field type	Translated to V12 field type	Notes
Text	String	Buffer size is same as Access field size
Number (byte)	Integer	
Number (integer)	Integer	
Number (long integer)	Integer	
Number (single)	Float	
Number (double)	Float	
Number (replication ID)	Ignored	
Currency	Integer	
Date / Time	Ignored	
Autonumber	Integer	
Yes/No	Integer	
OLE Object	Ignored	
HyperLink	String	URL imported as text
Memo	String	Buffer size is 32K

MS Access unique and duplicate indexes are properly converted to unique and duplicate INM V12 Database indexes with ascending field values.

Microsoft Excel

mReadDBstructure from MS Excel (Windows Only)

MS Excel workbooks can be used as templates to V12 databases. MS Excel workbooks can contain one or more worksheets, with each worksheet corresponding to a V12 table and each column to a V12 field. The resulting V12 database contains as many tables as there are worksheets in the Excel file. The MS Excel ODBC driver is required to perform this operation.

The names of the worksheets and columns of your MS Excel file must be valid V2-DBE identifiers (see Defining identifiers in Step 1: Decide on a Data Model).

The types of the field defined in the new V12 database depend on the format of the corresponding MS Excel columns. To change the format of a entire column in MS Excel, select it by clicking in its heading, choose Format > Cells... and select the Number tab. It may be necessary to Save As... your workbook with a new name to force MS Excel to commit to the new column's format (depends on version of Excel).

Syntax:

```
mReadDBStructure(gDB, "Excel", FileName)
```

where `FileName` is the path to the *.XLS file.

Importing from Microsoft Excel (Windows only)

MS Excel workbooks (*.XLS) can be imported to V12 databases, one table at a time, through a PC's ODBC driver. No DSN (Data Source Name) is required.

Syntax:

```
mImport(gTable, "Excel", FileName, TableName)
```

where:

- `FileName` is the path to the source *.XLS file. It is assumed to be encoded in the Windows ANSI character set (default encoding on Windows).
- `TableName` is the name of the table to import.

Example:

```
mImport(gTable, "Excel", the pahtname&"Results.XLS")
```

Protected MS Excel workbooks cannot be imported

Converting a MS Excel workbook into a V12 database is a two-step process: First, create the V12 database (see Importing from Microsoft Excel (Windows only)). Then, import data to each of its tables with `mImport`, as explained above.

Note: It is important that the first row contains the field names. This way, INM V12 Database can associate an Excel column to a INM V12 Database field.

Microsoft Excel Rules

The following rules apply to the translation of MS Excel file structures to V12 databases:

MS Excel field type	Translated to V12 field type	Notes
General	Float	
Number	Float	
Currency	Integer	
Accounting	Integer	
Date	Ignored	Convert to text first if importing to INM V12 Database is needed
Time	Ignored	Convert to text first if importing to INM V12 Database is needed
Percentage	Float	
Fraction	Float	
Scientific	Float	
Text	String	Buffer size = 255 bytes
Special	Float	
Custom	Float	

MS Excel cannot define indexes on its fields, when reading an Excel workbook; INM V12 Database automatically indexes the leftmost field of each worksheet.

Microsoft SQL Server

mReadDBstructure from MS SQL Server (Windows Only)

A MS SQL Server version 6 or 7 data source can be used as a template to a V12 database. In contrast to MS Access, MS FoxPro and MS Excel files, `mReadDBstructure` requires a DSN (Data Source Name) to be supplied instead of a pathname. The MS SQL Server ODBC driver is required to perform this operation.

Syntax:

```
mReadDBStructure(gDB, "SQLserver", DSN, Username, Password)
```

where

- `DSN` is the name of a valid User DSN, System DSN or File DSN (see Window's Control Panel),
    - `Username` is a valid user name to access the DSN,
    - `Password` is Username's matching password.

Importing from MS SQL (Windows only)

MS SQL Server data sources can be imported to V12 databases, one table at a time, through a PC's ODBC driver and a valid DSN (Data Source Name). Data sources can be created through Window's ODBC Data Sources Control Panel which is accessible from Start > Settings > Control Panel menu.

Syntax:

```
mImport(gTable, "SQLserver", DSN, Username, Password, TableName)
```

where:

- `DSN` is a valid Data Source Name.
- `Username` is a valid user name to access the SQL Server.
- `Password` is Username's matching password.
- `TableName` is the name of the table to import.

Example:

```
mImport(gTable, "SQLserver", "InventoryDSN", "Admin", "XBF48",
   "Products")
```

Converting an MS SQL Server data source into a V12 database is a two-step process: First, create the V12 database (see mReadDBstructure from MS SQL Server (Windows Only)). Then, import data to each of its tables with `mImport`, as explained above.

Microsoft SQL format Rules

The following rules apply to the translation of MS SQL Server data sources to V12 databases:

MS SQL Server field type	Translated to V12 field type	Notes
Binary	Ignored	
Bit	Integer	
Char	String	Buffer size is same as MS SQL Server field size
DateTime	Ignored	
Decimal	Float	
Float	Float	
Image	String	Buffer size = 32K. Data cannot be imported from Image fields.
Int	Integer	
Money	Float	
Numeric	Integer	
Real	Float	
SmallDateSize	Ignored	
SmallInt	Integer	
SmallMoney	Float	
SysName	String	Buffer size is same as MS SQL Server field size
Text	String	Buffer size = 32K
TimeStamp	Ignored	
TinyInt	Integer	
VarBinary	String	Buffer size is same as MS SQL Server field size
VarChar	String	Buffer size is same as MS SQL Server field size

Note that V12 does not support unicode SQL data type such as nchar, nvarchar and ntext.

Importing Media into a V12 Database

Before importing media into a V12 database, be sure to read the section on Using media with V12 Databases, which discusses alternative ways of dealing with media files.

You can import media to INM V12 Database fields of type `media`, one at a time, using the INM V12 Database Editor (see the INM V12 Database Editor User Manual).

You can also automate and customize the media importing process through Lingo scripting. Assume your database contains one table and five fields:

- Field `ItemName` of `type String`,
- Field `Description` of `type String`,
- Field `Price` of `type Float`,
- Field `CatalogNumber of` type `Integer`,
- Field `Photo` of type `Media`.

In addition, assume that the first four fields are in a TAB-delimited file format named "Data.txt", and that all photos (5th field) are in PICT format. Each photo is located in the same folder as "Data.txt" with each image file bearing the catalog number of the item with which it corresponds.

This example illustrates how to import the text file in a V12 Database file, and then how to review each imported record in order to import the corresponding image file.

Example:

```
-- some database creation preliminaries here
-- this is a purely academic example: no error trapping is
   performed
gDB = New(Xtra "V12dbe", the MoviePath&"Catalog.V12",
   "ReadWrite", "top secret")
gTable = New(Xtra "V12table", mGetRef(gDB), "Articles")

-- import the text data
mImport(gTable, "TEXT", the MoviePath&"Data.txt")

-- loop on each record and import the matching image
repeat with i = 1 to mSelectCount(gTable)

   -- record i becomes the current record
   mGo(gTable, i)

   -- get the photo's filename and import it in a member
   catNbr = mGetField(gTable, "catalogNumber")
   member("DummyMember").filename = (the MoviePath&catNbr)

   -- assign the photo to the appropriate INM V12 Database field
   mEditRecord(gTable)
```

```
        CheckV12Error()
        mSetMedia(gTable, "photo", member "DummyMember")
        CheckV12Error ()
        mUpdateRecord(gTable)
        CheckV12Error()
      end repeat

      gTable.mClose() -- close the table instance
      gTable = 0
      gDB.mClose()-- close the database instance
      gDB = 0
```

The `mAddRecord`, `mSetField`, `mGetField` and `mUpdateRecord` methods are explained in greater detail in *Using Lingo*. Refer to the INM V12 Database Methods Reference for explicit examples.

Appendix 2: mGetSelection Examples

The examples below show various ways of using `mGetSelection`. All examples assume that the table `gTable` contains 3 fields ("name", "price" and "number", declared in that order when creating the table), and that the selection contains 6 records.

Read an Entire Selection

This example retrieves the entire content of each record of the selection with TABs as field delimiters and CARRIAGE_RETURNS (CRs) as record delimiters. Fields are sorted in their order of creation. The records' sort order is the one defined by the selection.

```
x = mGetSelection(gTable)
```

sets the variable x to the following string:

```
Batteries   TAB   9.20   TAB   6780   CR
Floppies    TAB   1.89   TAB   9401   CR
Labels      TAB   1.19   TAB   1743   CR
Pencils     TAB   5.55   TAB   6251   CR
Ruler       TAB   1.99   TAB   1431   CR
Tags        TAB   6.19   TAB   7519   CR
```

Read a Range of Records in a String variable

This example retrieves the content of 3 successive records in the selection, starting with record #2, with TABs as field delimiters and CARRIAGE_RETURNS (CRs) as record delimiters.

```
x = mGetSelection(gTable, "LITERAL", 2, 3)
```

sets the variable x to the following string:

```
Floppies    TAB   1.89   TAB   9401   CR
Labels      TAB   1.19   TAB   1743   CR
Pencils     TAB   5.55   TAB   6251   CR
```

Read a Range of Records in a Lingo List

This is identical to the previous example, except that the result is returned in a Lingo list:

```
x = mGetSelection(gTable, "LIST", 2, 3)
```

sets the variable x to the following list:

```
[ ["Floppies", 1.89, 9401], ["Labels", 1.19, 1743], ["Pencils",
  5.55, 6251] ]
```

Read a Range of Records in a Property List

Same as the two previous examples, except that the result is returned in a Lingo property list:

```
x = mGetSelection(gTable, "PropertyList", 2, 3)
```

sets the variable x to the following list:

```
[ [#name:"Floppies", #price:1.89, #number:9401], [#name:"Labels",
  #price:1.19, #number:1743], [#name:"Pencils", #price:5.55,
  #number:6251] ]
```

Read the Entire Content of the Current Record

This example retrieves the entire content of the current record in a single call to INM V12 Database.

```
x = mGetSelection(gTable, "LITERAL", mGetPosition(gTable), 1)
```

sets the variable x to the following string:

```
Batteries  TAB   9.20   TAB   6780   CR
```

The "List" and "ProperyList" would respectively return:

```
[ ["Batteries", 9.20, 6780] ]
```

and

```
[ [#name:"Batteries", #price:9.20, #number:6780] ]
```

Read a Record without Setting it as the Current Record

This example retrieves the content of record #4 without setting it as the current record.

```
x = mGetSelection(gTable, "LITERAL", 4, 1)
```

sets the variable x to the following string:

```
Pencils   TAB   5.55   TAB   6251   CR
```

The "List" and "ProperyList" would respectively return:

```
[ ["Pencils", 5.55, 6251] ]
```

and

```
[ [#name:"Pencils",  #price:5.55, #number:6251] ]
```

Read the Entire Selection with Special Delimiters

This example retrieves the entire content of each record of the selection with commas (",") as field delimiters and slashes ("/") as record delimiters.

```
x = mGetSelection(gTable, "LITERAL", 1, mSelectCount(gTable), ","
  , "/" )
```

sets the variable x to the following string:

```
Batteries , 9.20 , 6780 / Floppies , 1.89 , 9401 / Labels , 1.19 ,
1743 / Pencils , 5.55 , 6251 / Ruler , 1.99 , 1431 / Tags , 6.19 ,
7519 /
```

Read Selected Fields in a Selection

This example retrieves the content of a single field ("name") for all the records of the selection. Note that the TAB parameter is unused in the result, but it should nonetheless be present.

```
x = mGetSelection(gTable, "LITERAL", 1, mSelectCount(gTable), TAB
   , RETURN, "name" )
```

sets the variable x to the following string:

```
Batteries  CR
Floppies   CR
Labels     CR
Pencils    CR
Ruler      CR
Tags       CR
```

The syntax for the Lingo List result would be:

```
x = mGetSelection(gTable, "List", 1, mSelectCount(gTable), "name"
   )
```

and the result would be

```
[["Batteries"],["Floppies"],["Labels"],["Pencils"],["Ruler"],
   ["Tags"]]
```

(Note: This is a list where each element is itself a single element list).

The syntax for the Property List result would be:

```
x = mGetSelection(gTable, "PropertyList", 1,
   mSelectCount(gTable), "name" )
```

and the result would be

```
[[#name:"Batteries"],[#name:"Floppies"],[#name:"Labels"],
   [#name:"Pencils"],[#name:"Ruler"],[#name:"Tags"]]
```

Read Records with a Determined Order of Fields

This example retrieves the content of all the records of the selection with TABs as field delimiters and CARRIAGE_RETURNS (CRs) as record delimiters, with fields ordered in the sequence "number", "name", and "price".

```
x = mGetSelection(gTable, "LITERAL", 1, mSelectCount(gTable),
   TAB, RETURN, "number", "name", "price")
```

sets the variable x to the following string:

```
6780   TAB   BatteriesTAB   9.20   CR
9401   TAB   Floppies TAB   1.89   CR
1743   TAB   Labels   TAB   1.19   CR
6251   TAB   Pencils  TAB   5.55   CR
1431   TAB   Ruler    TAB   1.99   CR
7519   TAB   Tags     TAB   6.19   CR
```

The syntax for the Lingo List result would be:

```
x = mGetSelection(gTable, "List", 1, mSelectCount(gTable),
   "number", "name", "price")
```

and the result would be

```
[ [6780, "Batteries", 9.20], [9401, "Floppies", 1.89], [1743,
  "Labels", 1.19], [6251, "Pencils", 5.55], [1431, "Ruler",
  1.99], [7519, "Tags", 6.19] ]
```

The syntax for the Property List result would be:

```
x = mGetSelection(gTable, "PropertyList", 1,
  mSelectCount(gTable), "number", "name", "price")
```

and the result would be

```
[ [#number:6780, #name:"Batteries", #price:9.20], [#number:9401,
  #name:"Floppies", #price:1.89], [#number:1743, #name:"Labels",
  #price:1.19], [#number:6251, #name:"Pencils", #price:5.55],
  [#number:1431, #name:"Ruler", #price:1.99], [#number:7519,
  #name:"Tags", #price:6.19] ]
```

Although, this latter request would not be of much interest because property lists are parsed by property names, not item positions.

Appendix 3: String and Custom String Types

INM V12 Database enables you to develop applications containing different types of strings such as English, German, Swedish and Spanish. Basically, each INM V12 Database table can contain any combination of those string types.

String comparisons depend on how special characters are defined in their corresponding languages. For example, the letters **a** and **ä** may be considered identical in some languages but different in others. This behavior is determined by the *sorting and searching rules* attached to each type of string.

INM V12 Database's default and custom String types' sorting and searching rules are defined by the following tables where equivalent characters are listed on the same line separated by one or more spaces and strict precedence is indicated by characters on successive lines. For example:

```
j J
k K
l L
```

means that:

- K sorts after J and before L,
- j and J are equivalent (likewise, k and K are equivalent, and l and L are equivalent too)

Characters omitted from a sorting and searching rules table are considered to sort *after* those listed in the table, except for Control characters (such as Carriage Return, Horizontal Tab, Vertical Tab, etc.), which are considered to sort *before* those listed in the table.

The Default String

The default `string` type has predefined rules that accommodate a large number of languages (English, French, German, Italian, Dutch, Portuguese, Norwegian, etc.).

(If the tables below are not properly formatted in the HTML version of this manual, please refer to the PDF version)

Sorting and searching rules table for the default string type:

1.	' ‘ ’
2.	" « » “ ”
3.	! ¡
4.	? ¿
5.	.
6.	,
7.	:
8.	;
9.	…
10.	#
11.	$
12.	¢
13.	£
14.	¥
15.	% ‰
16.	°
17.	|
18.	† ‡
19.	[ ]
20.	{ }
21.	( )
22.	< >
23.	*
24.	+
25.	-
26.	/
27.	\
28.	=
29.	~
30.	¬ - – —
31.	§
32.	µ
33.	&
34.	@
35.	©
36.	ƒ
37.	®
38.	0
39.	1
40.	2
41.	3
42.	4
43.	5
44.	6
45.	7
46.	8
47.	9
48.	a à á â ã ä A À Á Â Ã Ä
49.	b B
50.	c ç C Ç
51.	d D
52.	e è é ê ë E È É Ê Ë
53.	f F
54.	g G
55.	h H
56.	i ì í î ï I Ì Í Î Ï
57.	j J
58.	k K
59.	l L
60.	m M
61.	n ñ N Ñ
62.	o ò ó ô õ ö œ O Ò Ó Ô Õ Ö Œ
63.	p P
64.	q Q
65.	r R
66.	s ß S
67.	t T
68.	u ù ú û ü U Ù Ú Û Ü
69.	v V
70.	w W
71.	x X
72.	y ÿ Y Ÿ
73.	z Z

74.	æ Æ
75.	ø Ø
76.	å Å

Predefined Custom String Types

Along with the standard `string` type, INM V12 Database contains a number of predefined custom string types. They include `Swedish`, `Spanish` and `Hebrew`.

Searching and Sorting rules for Strings of Type *Swedish*

(If the tables below are not properly formatted in this version of this manual, please refer to the PDF version)

1.	' ‘ ’	40.	2
2.	" « » “ ”	41.	3
3.	! ¡	42.	4
4.	? ¿	43.	5
5.	.	44.	6
6.	,	45.	7
7.	:	46.	8
8.	;	47.	9
9.	…	48.	a à á â ã A À Á Â Ã
10.	#	49.	b B
11.	$	50.	c ç C Ç
12.	¢	51.	d D
13.	£	52.	e è é ê ë E È É Ê Ë
14.	¥	53.	f F
15.	% ‰	54.	g G
16.	°	55.	h H
17.	|	56.	i ì í î ï I Ì Í Î Ï
18.	† ‡	57.	j J
19.	[ ]	58.	k K
20.	{ }	59.	l L
21.	( )	60.	m M
22.	< >	61.	n ñ N Ñ
23.	*	62.	o ò ó ô õ œ O Ò Ó Ô Õ Œ
24.	+	63.	p P
25.	-	64.	q Q
26.	/	65.	r R
27.	\	66.	s ß S
28.	=	67.	t T
29.	~	68.	u ù ú û ü U Ù Ú Û Ü
30.	¬ - – —	69.	v V
31.	§	70.	w W
32.	µ	71.	x X
33.	&	72.	y ÿ Y Ÿ
34.	@	73.	z Z
35.	©	74.	æ Æ

36.	ƒ	75.	ø Ø
37.	®	76.	å Å
38.	0	77.	ä Ä
39.	1	78.	ö Ö

Searching and Sorting rules for Strings of Type *Spanish*

(If the tables below are not properly formatted in this version of this manual, please refer to the PDF version)

1.	' ‘ ’
2.	" « » “ ”
3.	! ¡
4.	? ¿
5.	.
6.	,
7.	:
8.	;
9.	…
10.	#
11.	$
12.	¢
13.	£
14.	¥
15.	% ‰
16.	°
17.	|
18.	† ‡
19.	[ ]
20.	{ }
21.	( )
22.	< >
23.	*
24.	+
25.	-
26.	/
27.	\
28.	=
29.	~
30.	¬ ‐ – —
31.	§
32.	µ
33.	&
34.	@
35.	©
36.	ƒ
37.	®
38.	0
39.	1
40.	2
41.	3
42.	4
43.	5
44.	6
45.	7
46.	8
47.	9
48.	a à á â ã ä A À Á Â Ã Ä
49.	b B
50.	c ç C Ç
51.	d D
52.	e è é ê ë E È É Ê Ë
53.	f F
54.	g G
55.	h H
56.	i ì í î ï I Ì Í Î Ï
57.	j J
58.	k K
59.	l L
60.	m M
61.	n N
62.	ñ Ñ
63.	o ò ó ô õ ö œ O Ò Ó Ô Õ Ö Œ
64.	p P
65.	q Q
66.	r R
67.	s ß S
68.	t T
69.	u ù ú û ü U Ù Ú Û Ü
70.	v V
71.	w W
72.	x X
73.	y ÿ Y Ÿ
74.	z Z
75.	æ Æ
76.	ø Ø
77.	å Å

Searching and Sorting rules for Strings of Type *Hebrew*

(requires a Hebrew font such as "Web Hebrew")

1.	' ' ‘	
2.	" « » ” “	
3.	! ¡	
4.	? ¿	
5.	.	
6.	,	
7.	:	
8.	;	
9.	…	
10.	#	
11.	$	
12.	¢	
13.	£	
14.	¥	
15.	% ‰	
16.	°	
17.	|	
18.	† ‡	
19.	[ ]	
20.	{ }	
21.	( )	
22.	< >	
23.	*	
24.	+	
25.	-	
26.	/	
27.	\	
28.	=	
29.	~	
30.	- – ¬	
31.	§	
32.	µ	
33.	&	
34.	@	
35.	©	
36.	ƒ	
37.	®	
38.	à	(aleph)
39.	á	(beth)
40.	â	(ghimel)
41.	ã	(daleth)
42.	ä	(he)
43.	å	(vau)
50.	í î	(mem)
51.	ï ð	(nun)
52.	ñ	(samech)
53.	ò	(ain)
54.	ó ô	(phe)
55.	õ ö	(sadi)
56.	÷	(koph)
57.	ø	(resch)
58.	ù	(sin)
59.	ú	(tau)
60.	0	
61.	1	
62.	2	
63.	3	
64.	4	
65.	5	
66.	6	
67.	7	
68.	8	
69.	9	
70.	a A À Á Â Ã Ä	
71.	b B	
72.	c C Ç	
73.	d D	
74.	e E È É Ê Ë	
75.	f F	
76.	g G	
77.	h H	
78.	i I Ì Í Î Ï	
79.	j J	
80.	k K	
81.	l L	
82.	m M	
83.	n N Ñ	
84.	o O Ò Ó Ô Õ Ö	
85.	p P	
86.	q Q	
87.	r R	
88.	s ß S	
89.	t T	
90.	u û ü U Ù Ú Û Ü	
91.	v V	
92.	w W	

44.	æ	(zain)	93. x X
45.	ç	(heth)	94. y ÿ Y Ÿ
46.	è	(teth)	95. z Z
47.	é	(iod)	96. Æ
48.	ê ë	(caph)	97. Ø
49.	ì	(lamed)	98. Å

Appendix 4: Character sets

Windows-ANSI Character Set

32		70	F	108	l	146	’	184	¸	222	Þ
33	!	71	G	109	m	147	“	185	¹	223	ß
34	"	72	H	110	n	148	”	186	º	224	à
35	#	73	I	111	o	149	•	187	»	225	á
36	$	74	J	112	p	150	–	188	¼	226	â
37	%	75	K	113	q	151	—	189	½	227	ã
38	&	76	L	114	r	152	˜	190	¾	228	ä
39	'	77	M	115	s	153	™	191	¿	229	å
40	(	78	N	116	t	154	š	192	À	230	æ
41	)	79	O	117	u	155	›	193	Á	231	ç
42	*	80	P	118	v	156	œ	194	Â	232	è
43	+	81	Q	119	w	157		195	Ã	233	é
44	,	82	R	120	x	158		196	Ä	234	ê
		83	S	121	y	159	Ÿ	197	Å	235	ë
		84	T	122	z	160		198	Æ	236	ì
		85	U	123	{	161	¡	199	Ç	237	í
		86	V	124	|	162	¢	200	È	238	î
		87	W	125	}	163	£	201	É	239	ï
		88	X	126	~	164	¤	202	Ê	240	ð
		89	Y	127		165	¥	203	Ë	241	ñ
		90	Z	128		166	¦	204	Ì	242	ò
		91	[	129		167	§	205	Í	243	ó
		92	\	130	‚	168	¨	206	Î	244	ô
		93	]	131	ƒ	169	©	207	Ï	245	õ
		94	^	132	„	170	ª	208	Ð	246	ö
		95	_	133	…	171	«	209	Ñ	247	÷
		96	`	134	†	172	¬	210	Ò	248	ø
		97	a	135	‡	173	-	211	Ó	249	ù
		98	b	136	ˆ	174	®	212	Ô	250	ú
		99	c	137	‰	175	¯	213	Õ	251	û
		100	d	138	Š	176	°	214	Ö	252	ü
		101	e	139	‹	177	±	215	×	253	ý
		102	f	140	Œ	178	²	216	Ø	254	þ
		103	g	141		179	³	217	Ù	255	ÿ
		104	h	142		180	´	218	Ú		
		105	i	143		181	µ	219	Û		
		106	j	144		182	¶	220	Ü		
		107	k	145	‘	183	·	221	Ý		

4
5

-

4
6

.

4
7

/

4
8

0

4
9

1

5
0

2

5
1

3

5
2

4

5
3

5

5
4

6

5
5

7

5
6

8

5
7

9

5
8

:

5
9

	;
60	**<**
61	**=**
62	**>**
63	**?**
64	@
65	**A**
66	**B**
67	**C**
68	**D**
69	**E**

Mac-Standard Character Set

32		70	F	108	l	146	í	184	Þ	222	
33	!	71	G	109	m	147	ì	185	þ	223	
34	"	72	H	110	n	148	î	186	Š	224	‡
35	#	73	I	111	o	149	ï	187	ª	225	·
36	$	74	J	112	p	150	ñ	188	º	226	‚
37	%	75	K	113	q	151	ó	189	ý	227	„
38	&	76	L	114	r	152	ò	190	æ	228	‰
39	'	77	M	115	s	153	ô	191	ø	229	Â
40	(	78	N	116	t	154	ö	192	¿	230	Ê
41	)	79	O	117	u	155	õ	193	¡	231	Á
42	*	80	P	118	v	156	ú	194	¬	232	Ë
43	+	81	Q	119	w	157	ù	195	¯	233	È
44	,	82	R	120	x	158	û	196	ƒ	234	Í
45	-	83	S	121	y	159	ü	197	¼	235	Î
46	.	84	T	122	z	160	†	198	Ð	236	Ï
47	/	85	U	123	{	161	°	199	«	237	Ì
48	0	86	V	124	|	162	¢	200	»	238	Ó
49	1	87	W	125	}	163	£	201	…	239	Ô
50	2	88	X	126	~	164	§	202		240	
51	3	89	Y	127		165	•	203	À	241	Ò
52	4	90	Z	128	Ä	166	¶	204	Ã	242	Ú
53	5	91	[	129	Å	167	ß	205	Õ	243	Û
54	6	92	\	130	Ç	168	®	206	Œ	244	Ù
55	7	93	]	131	É	169	©	207	œ	245	¦
56	8	94	^	132	Ñ	170	™	208	-	246	ˆ
57	9	95	_	133	Ö	171	´	209	—	247	˜
58	:	96	`	134	Ü	172	¨	210	“	248	–
59	;	97	a	135	á	173		211	”	249	š
60	<	98	b	136	à	174	Æ	212	‘	250	²
61	=	99	c	137	â	175	Ø	213	’	251	¾
62	>	100	d	138	ä	176		214	÷	252	¸
63	?	101	e	139	ã	177	±	215	×	253	½
64	@	102	f	140	å	178		216	ÿ	254	³
65	A	103	g	141	ç	179		217	Ÿ	255	¹
66	B	104	h	142	é	180	¥	218			
67	C	105	i	143	è	181	µ	219	¤		
68	D	106	j	144	ê	182	ð	220	‹		
69	E	107	k	145	ë	183	Ý	221	›		

MS-DOS Character Set

32		70	F	108	l	146	Æ	184	©	222	Ì
33	!	71	G	109	m	147	ô	185	¦	223	_
34	"	72	H	110	n	148	ö	186	¦	224	Ó
35	#	73	I	111	o	149	ò	187	+	225	ß
36	$	74	J	112	p	150	û	188	+	226	Ô
37	%	75	K	113	q	151	ù	189	¢	227	Ò
38	&	76	L	114	r	152	ÿ	190	¥	228	õ
39	'	77	M	115	s	153	Ö	191	+	229	Õ
40	(	78	N	116	t	154	Ü	192	+	230	µ
41	)	79	O	117	u	155	ø	193	-	231	þ
42	*	80	P	118	v	156	£	194	-	232	Þ
43	+	81	Q	119	w	157	Ø	195	+	233	Ú
44	,	82	R	120	x	158	×	196	-	234	Û
45	-	83	S	121	y	159	ƒ	197	+	235	Ù
46	.	84	T	122	z	160	á	198	ã	236	ý
47	/	85	U	123	{	161	í	199	Ã	237	Ý
48	0	86	V	124	|	162	ó	200	+	238	¯
49	1	87	W	125	}	163	ú	201	+	239	´
50	2	88	X	126	~	164	ñ	202	-	240	-
51	3	89	Y	127		165	Ñ	203	-	241	±
52	4	90	Z	128	Ç	166	ª	204	¦	242	_
53	5	91	[	129	ü	167	º	205	-	243	¾
54	6	92	\	130	é	168	¿	206	+	244	¶
55	7	93	]	131	â	169	®	207	¤	245	§
56	8	94	^	132	ä	170	¬	208	ð	246	÷
57	9	95	_	133	à	171	½	209	Ð	247	¸
58	:	96	`	134	å	172	¼	210	Ê	248	°
59	;	97	a	135	ç	173	¡	211	Ë	249	¨
60	<	98	b	136	ê	174	«	212	È	250	·
61	=	99	c	137	ë	175	»	213	i	251	¹
62	>	100	d	138	è	176	_	214	Í	252	³
63	?	101	e	139	ï	177	_	215	Î	253	²
64	@	102	f	140	î	178	_	216	Ï	254	_
65	A	103	g	141	ì	179	¦	217	+	255	
66	B	104	h	142	Ä	180	¦	218	+		
67	C	105	i	143	Å	181	Á	219	_		
68	D	106	j	144	É	182	Â	220	_		
69	E	107	k	145	æ	183	À	221	¦		

Appendix 5: Japanese support

New field types

INM V12 Database supports Japanese text by adding two new field types: String.SJIS and String.YOMI. The mReadDBStructure method must be used to create those two new fields.

Example:

```
[TABLE]
NameOfTable
[FIELDS]
sjisfield string.SJIS
yomifield string.YOMI
[INDEXES]
sjisfieldndx duplicate sjisfield ascending
yomifieldndx duplicate yomifield ascending
[END]
```

Field of type SJIS

SJIS fields can hold Japanese text in the Shift-JIS format. Katakana, Hiragana, English alphabets, punctuation, numerals and Kanji are available in this representation.

Sorting

A SJIS sort will order the fields (thus the records) according to the Shift-JIS numerical code of the characters (1 or 2 bytes) in it. There is no strict equivalence of characters in this sort order. This means that each character has a distinct location in the table and will always sort the same way in a similar sort operation.

Note: The SJIS field does not support full text indexing. The following operators are not supported: wordStarts and wordEquals.

Searching

All operators are supported and Boolean set operators AND and OR are also supported.

Field of type Yomi (Yomigana)

This field can hold a subset of Shift-JIS character. This subset is restricted to the full range of Katakana, hiragana including those with voiced diacritic marks (nigori, maru) and small characters (contracted sounds). It can also contain punctuation characters. This field doesn’t contain kanji.

Sorting

The exact sorting order of the syllabaries is given by the following table:

Phonetic characters (hiragana and katakana)	Pronunciation (each sounds corresponds to 2 characters in the left column)
ぁァあアぃィいイぅゥうウぇェえエぉォおオ	(a) a (i) i (u) u (e) e (o) o
かカがガきキぎギくクぐグけケげゲこコごゴ	ka ga ki gi ku gu ke ge ko go
さサざザしシじジすスずズせセぜゼそソぞゾ	sa za shi ji su zu se ze so zo
たタだダちチぢヂっッつツづヅてテでデとトどド	ta da chi dzi (tsu) tsu dzu te de to do
なナにニぬヌねネのノ	na ni nu ne no
はハばバぱパひヒびビぴピふフぶブぷプへヘべベぺペほホぼボぽポ	ha ba pa hi bi pi fu bu pu he be pe ho bo po
まマみミむムめメもモ	ma mi mu me mo
ゃャやヤゅュゆユょョよヨ	(ya) ya (yu) yu (yo) yo
らラりリるルれレろロ	ra ri ru re ro
わワをヲんン	wa wo n

Note: requires a Japanese font such as "MS Mincho".

Dash symbol

The dash symbol has a special meaning in Yomigana. This special treatment breaks the uniformity of the comparison method of the sorting procedure. To remove this problem, the dash symbol should be replaced by its respective vowel in the input data. In other words, the compare method does not take into account the semantic of the dash symbol. The mSetCriteria method translates the dash symbol to the correct (preceding) vowel. The translation is defined in the following table.

Reading	Kana	Dashed
1/2 a (hiragana)	ぁ	ぁあ
1/2 a (katakana)	ァ	ァア
a (hiragana)	あ	ああ
a (katakana)	ア	アア
1/2 i (hiragana)	ぃ	ぃい
1/2 i (katakana)	ィ	ィイ
i (hiragana)	い	いい
i (katakana)	イ	イイ
1/2 u (hiragana)	ぅ	ぅう
1/2 u (katakana)	ゥ	ゥウ
u (hiragana)	う	うう
u (katakana)	ウ	ウウ
1/2 e (hiragana)	ぇ	ぇえ
1/2 e (katakana)	ェ	ェエ
e (hiragana)	え	ええ
e (katakana)	エ	エエ
1/2 o (hiragana)	ぉ	ぉお
1/2 o (katakana)	ォ	ォオ
o (hiragana)	お	おお
o (katakana)	オ	オオ

ka (hiragana)	か	かあ
ka (katakana)	カ	カア
ga (hiragana)	が	があ
ga (katakana)	ガ	ガア
ki (hiragana)	き	きい
ki (katakana)	キ	キイ
gi (hiragana)	ぎ	ぎい
gi (katakana)	ギ	ギイ
ku (hiragana)	く	くう
ku (katakana)	ク	クウ
gu (hiragana)	ぐ	ぐう
gu (katakana)	グ	グウ
ke (hiragana)	け	けえ
ke (katakana)	ケ	ケエ
ge (hiragana)	げ	げえ
ge (katakana)	ゲ	ゲエ
ko (hiragana)	こ	こお
ko (katakana)	コ	コオ
go (hiragana)	ご	ごお
go (katakana)	ゴ	ゴオ
sa (hiragana)	さ	さあ

sa (katakana)	サ	サア
za (hiragana)	ざ	ざあ
za (katakana)	ザ	ザア
shi (hiragana)	し	しい
shi (katakana)	シ	シイ
ji (hiragana)	じ	じい
ji (katakana)	ジ	ジイ
su (hiragana)	す	すう
su (katakana)	ス	スウ
zu (hiragana)	ず	ずう
zu (katakana)	ズ	ズウ
se (hiragana)	せ	せえ
se (katakana)	セ	セエ
ze (hiragana)	ぜ	ぜえ
ze (katakana)	ゼ	ゼエ
so (hiragana)	そ	そお
so (katakana)	ソ	ソオ
zo (hiragana)	ぞ	ぞお
zo (katakana)	ゾ	ゾオ
ta (hiragana)	た	たあ
ta (katakana)	タ	タア

da (hiragana)	だ	だあ
da (katakana)	ダ	ダア
chi (hiragana)	ち	ちい
chi (katakana)	チ	チイ
dzi (hiragana)	ぢ	ぢい
dzi (katakana)	ヂ	ヂイ
1/2 tsu (hiragana)	っ	っう
1/2 tsu (katakana)	ッ	ッウ
tsu (hiragana)	つ	つう
tsu (katakana)	ツ	ツウ
dzu (hiragana)	づ	づう
dzu (katakana)	ヅ	ヅウ
te (hiragana)	て	てえ
te (katakana)	テ	テエ
de (hiragana)	で	でえ
de (katakana)	デ	デエ
to (hiragana)	と	とお
to (katakana)	ト	トオ
do (hiragana)	ど	どお
do (katakana)	ド	ドオ
na (hiragana)	な	なあ

na (katakana)	ナ	ナア
ni (hiragana)	に	にい
ni (katakana)	ニ	ニイ
nu (hiragana)	ぬ	ぬう
nu (katakana)	ヌ	ヌウ
ne (hiragana)	ね	ねえ
ne (katakana)	ネ	ネエ
no (hiragana)	の	のお
no (katakana)	ノ	ノオ
ha (hiragana)	は	はあ
ha (katakana)	ハ	ハア
ba (hiragana)	ば	ばあ
ba (katakana)	バ	バア
pa (hiragana)	ぱ	ぱあ
pa (katakana)	パ	パア
hi (hiragana)	ひ	ひい
hi (katakana)	ヒ	ヒイ
bi (hiragana)	び	びい
bi (katakana)	ビ	ビイ
pi (hiragana)	ぴ	ぴい
pi (katakana)	ピ	ピイ

fu (hiragana)	ふ	ふう
fu (katakana)	フ	フウ
bu (hiragana)	ぶ	ぶう
bu (katakana)	ブ	ブウ
pu (hiragana)	ぷ	ぷう
pu (katakana)	プ	プウ
he (hiragana)	へ	へえ
he (katakana)	ヘ	ヘエ
be (hiragana)	べ	べえ
be (katakana)	ベ	ベエ
pe (hiragana)	ぺ	ぺえ
pe (katakana)	ペ	ペエ
ho (hiragana)	ほ	ほお
ho (katakana)	ホ	ホオ
bo (hiragana)	ぼ	ぼお
bo (katakana)	ボ	ボオ
po (hiragana)	ぽ	ぽお
po (katakana)	ポ	ポオ
ma (hiragana)	ま	まあ
ma (katakana)	マ	マア
mi (hiragana)	み	みい

mi (katakana)	ミ	ミイ
mu (hiragana)	む	むう
mu (katakana)	ム	ムウ
me (hiragana)	め	めえ
me (katakana)	メ	メエ
mo (hiragana)	も	もお
mo (katakana)	モ	モオ
1/2 ya (hiragana)	ゃ	ゃあ
1/2 ya (katakana)	ャ	ャア
ya (hiragana)	や	やあ
ya (katakana)	ヤ	ヤア
1/2 yu (hiragana)	ゅ	ゅう
1/2 yu (katakana)	ュ	ュウ
yu (hiragana)	ゆ	ゆう
yu (katakana)	ユ	ユウ
1/2 yo (hiragana)	ょ	ょお
1/2 yo (katakana)	ョ	ョオ
yo (hiragana)	よ	よお
yo (katakana)	ヨ	ヨオ
ra (hiragana)	ら	らあ
ra (katakana)	ラ	ラア

ri (hiragana)	り	りい
ri (katakana)	リ	リイ
ru (hiragana)	る	るう
ru (katakana)	ル	ルウ
re (hiragana)	れ	れえ
re (katakana)	レ	レエ
ro (hiragana)	ろ	ろお
ro (katakana)	ロ	ロオ
wa (hiragana)	わ	わあ
wa (katakana)	ワ	ワア
wo (hiragana)	を	をお
wo (katakana)	ヲ	ヲオ
n (hiragana)	ん	n/a
n (katakana)	ン	n/a
dash	ー	n/a

Note: The last 3 entries (n, n, dash) wouldn't be followed by a dash in Japanese text, so those can be ignored for the purposes of dash-ing

Searching

The Yomi field has exactly the same search characteristics as the SJIS field. See the details under Fields of Type SJIS, Searching.

Data Importation

V12-J import Japanese text through the mImport method. No validation is done on the incoming data. It is up to the user to ensure that the text is valid. Fields and record delimiters remain 1-byte characters.

Index