



IBM

# **InterCall Developer's Guide**

Version 10.3  
February, 2009

IBM Corporation  
555 Bailey Avenue  
San Jose, CA 95141

Licensed Materials – Property of IBM

© Copyright International Business Machines Corporation 2008, 2009. All rights reserved.

AIX, DB2, DB2 Universal Database, Distributed Relational Database Architecture, NUMA-Q, OS/2, OS/390, and OS/400, IBM Informix®, C-ISAM®, Foundation.2000™, IBM Informix® 4GL, IBM Informix® DataBlade® module, Client SDK™, Cloudscape™, Cloudsync™, IBM Informix® Connect, IBM Informix® Driver for JDBC, Dynamic Connect™, IBM Informix® Dynamic Scalable Architecture™ (DSA), IBM Informix® Dynamic Server™, IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager), IBM Informix® Extended Parallel Server™, i.Financial Services™, J/Foundation™, MaxConnect™, Object Translator™, Red Brick® Decision Server™, IBM Informix® SE, IBM Informix® SQL, InformiXML™, RedBack®, SystemBuilder™, U2™, UniData®, UniVerse®, wIntegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

This product includes cryptographic software written by Eric Young (eay@cryptosoft.com).

This product includes software written by Tim Hudson (tjh@cryptosoft.com).

Documentation Team: Claire Gustafson, Shelley Thompson, Anne Waite

#### US GOVERNMENT USERS RESTRICTED RIGHTS

Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Table of Contents

## Preface

Organization of This Manual . . . . .	viii
Documentation Conventions. . . . .	ix
Help . . . . .	xi
API Documentation . . . . .	xii

## Chapter 1

### Introduction

About InterCall. . . . .	1-2
Minimum System Requirements . . . . .	1-3
InterCall Installation . . . . .	1-4
On Windows Platforms. . . . .	1-4
On UNIX Systems . . . . .	1-5
How InterCall Works . . . . .	1-6
Copying the Software. . . . .	1-7
UniVerse NLS in Client Programs . . . . .	1-8
NLS Configurable Parameters . . . . .	1-8
Character Mapping . . . . .	1-8
The Sample Program . . . . .	1-10

## Chapter 2

### Programming with InterCall

Server Sessions. . . . .	2-3
Using the Microsoft Security Token. . . . .	2-4
Argument Passing Conventions. . . . .	2-5
The ICSTRING Type . . . . .	2-5
Calling Functions from C Programs. . . . .	2-5
Calling Functions from Visual Basic Programs . . . . .	2-6
Using the @TTY Variable . . . . .	2-7

## Chapter 3

### InterCall Functions

Function Summary. . . . .	3-5
---------------------------	-----

Error Codes . . . . .	3-8
ic_alpha . . . . .	3-9
ic_calloc . . . . .	3-10
ic_cleardata . . . . .	3-11
ic_clearfile . . . . .	3-12
ic_clearselect . . . . .	3-14
ic_close . . . . .	3-16
ic_closeseq . . . . .	3-18
ic_data . . . . .	3-20
ic_date . . . . .	3-22
ic_delete . . . . .	3-23
ic_execute . . . . .	3-25
ic_executecontinue . . . . .	3-27
ic_extract . . . . .	3-29
ic_fileinfo . . . . .	3-31
ic_filelock . . . . .	3-34
ic_fileunlock . . . . .	3-36
ic_fmt . . . . .	3-38
ic_formlist . . . . .	3-40
ic_free . . . . .	3-42
ic_getlist . . . . .	3-43
ic_get_locale . . . . .	3-45
ic_get_map . . . . .	3-47
ic_get_mark_value . . . . .	3-49
ic_getvalue . . . . .	3-51
ic_iconv . . . . .	3-54
ic_indices . . . . .	3-56
ic_inputreply . . . . .	3-59
ic_insert . . . . .	3-61
ic_itype . . . . .	3-63
ic_locate . . . . .	3-65
ic_lock . . . . .	3-67
ic_lower . . . . .	3-68
ic_malloc . . . . .	3-70
ic_oconv . . . . .	3-71
ic_open . . . . .	3-73
ic_openseq . . . . .	3-76
ic_opensession . . . . .	3-78
ic_quit . . . . .	3-81
ic_quitall . . . . .	3-82
ic_raise . . . . .	3-83
ic_read . . . . .	3-85

ic_readblk . . . . .	3-88
ic_readlist . . . . .	3-90
ic_readnext . . . . .	3-92
ic_readseq . . . . .	3-94
ic_readv . . . . .	3-96
ic_recordlock . . . . .	3-99
ic_recordlocked . . . . .	3-102
ic_release . . . . .	3-105
ic_remove . . . . .	3-107
ic_replace . . . . .	3-110
ic_seek . . . . .	3-113
ic_select . . . . .	3-115
ic_selectindex . . . . .	3-117
ic_session_info . . . . .	3-119
ic_set_comms_timeout . . . . .	3-121
ic_set_locale . . . . .	3-123
ic_set_map (UniVerse only) . . . . .	3-125
ic_setsession . . . . .	3-127
ic_setvalue . . . . .	3-129
ic_strdel . . . . .	3-131
ic_subcall . . . . .	3-133
ic_time . . . . .	3-135
ic_timedate . . . . .	3-136
ic_trans . . . . .	3-138
ic_unlock . . . . .	3-140
ic_unidata_session . . . . .	3-141
ic_universe_session . . . . .	3-144
ic_weofseq . . . . .	3-147
ic_write . . . . .	3-149
ic_writeblk . . . . .	3-151
ic_writeseq . . . . .	3-153
ic_writelv . . . . .	3-155

## Appendix A    **InterCall Functions by Use**

Accessing a Server . . . . .	A-2
Reading and Modifying Records . . . . .	A-3
Reading and Modifying Sequential Files . . . . .	A-4
Accessing and Modifying Strings . . . . .	A-5
Accessing and Modifying Select Lists . . . . .	A-6
Managing Database Files . . . . .	A-7
Using UniVerse NLS (UniVerse Only). . . . .	A-8
Using System Utilities. . . . .	A-9

**Appendix B    Error Codes**

Database Error Codes . . . . . B-2

# Preface

This manual describes how to use InterCall, an application programming interface to UniVerse and UniData. This book is for application developers who have a good working knowledge of UniVerse or UniData, UNIX, and Windows environments.

---

## Organization of This Manual

This manual contains the following:

Chapter 1, “[Introduction](#),” introduces InterCall.

Chapter 2, “[Programming with InterCall](#),” describes programming with InterCall.

Chapter 3, “[InterCall Functions](#),” describes the InterCall functions.

Appendix A, “[InterCall Functions by Use](#),” contains a summary of InterCall functions listed by usage.

Appendix B, “[Error Codes](#),” describes the InterCall error reporting system and lists the error codes and their meanings.

The [Glossary](#) defines terms that are used in this manual.



---

## Documentation Conventions

This manual uses the following conventions:

Convention	Usage
<b>Bold</b>	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates UniVerse commands, keywords, and options; UniVerse BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates UniVerse identifiers such as file names, account names, schema names, and Windows file names and paths.
<i>Italic</i>	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, file names, and paths.
Courier	Courier indicates examples of source code and system output.
<b>Courier Bold</b>	In examples, courier bold indicates characters that the user types or keys the user presses (for example, <b>&lt;Return&gt;</b> ).
[ ]	Brackets enclose optional items. Do not type the brackets unless indicated.
{ }	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
itemA   itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
...	Three periods indicate that more of the same type of item can optionally follow.
->	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose <b>File</b> -> <b>Exit</b> ” means you should choose <b>File</b> from the menu bar, then choose <b>Exit</b> from the File menu.
¶	Item mark. For example, the item mark (¶) in the following string delimits elements 1 and 2, and elements 3 and 4: 1¶2¶3¶4¶5

---

### Documentation Conventions

Convention	Usage
<b>F</b>	Field mark. For example, the field mark ( <b>F</b> ) in the following string delimits elements FLD1 and VAL1: FLD1 <b>F</b> VAL1 <b>V</b> SUBV1 <b>S</b> SUBV2
<b>V</b>	Value mark. For example, the value mark ( <b>V</b> ) in the following string delimits elements VAL1 and SUBV1: FLD1 <b>F</b> VAL1 <b>V</b> SUBV1 <b>S</b> SUBV2
<b>S</b>	Subvalue mark. For example, the subvalue mark ( <b>S</b> ) in the following string delimits elements SUBV1 and SUBV2: FLD1 <b>F</b> VAL1 <b>V</b> SUBV1 <b>S</b> SUBV2
<b>T</b>	Text mark. For example, the text mark ( <b>T</b> ) in the following string delimits elements 4 and 5: 1 <b>F</b> 2 <b>S</b> 3 <b>V</b> 4 <b>T</b> 5

**Documentation Conventions (Continued)**

The following are also used:

- Syntax definitions and examples are indented for ease in reading.
- All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.
- Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

---

## Help

To get Help about InterCall, choose **Programs -> IBM U2 -> UniDK -> InterCall -> Help** from the **Start** menu.

---

## API Documentation

The following books document application programming interfaces (APIs) used for developing client applications that connect to UniVerse and UniData servers.

***Administrative Supplement for APIs:*** Introduces IBM's seven common APIs for UniData and UniVerse, and provides important information that developers using any of the common APIs will need. It includes information about the UniRPC, the UCI Config Editor, the *ud\_database* file, and device licensing.

***UCI Developer's Guide:*** Describes how to use UCI (Uni Call Interface), an interface to UniVerse and UniData databases from C-based client programs. UCI uses ODBC-like function calls to execute SQL statements on local or remote UniVerse and UniData servers. This book is for experienced SQL programmers.

***IBM JDBC Driver for UniData and UniVerse:*** Describes UniJDBC, an interface to UniData and UniVerse databases from JDBC applications. This book is for experienced programmers and application developers who are familiar with UniData and UniVerse, Java, JDBC, and who want to write JDBC applications that access these databases.

***InterCall Developer's Guide:*** Describes how to use the InterCall API to access data on UniVerse and UniData systems from external programs. This book is for experienced programmers who are familiar with UniVerse or UniData.

***UniObjects Developer's Guide:*** Describes UniObjects, an interface to UniVerse and UniData systems from Visual Basic. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Visual Basic, and who want to write Visual Basic programs that access these databases.

***UniObjects for Java Developer's Guide:*** Describes UniObjects for Java, an interface to UniVerse and UniData systems from Java. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Java, and who want to write Java programs that access these databases.

***UniObjects for .NET Developer's Guide:*** Describes UniObjects, an interface to UniVerse and UniData systems from .NET. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with .NET, and who want to write .NET programs that access these databases.

***Using UniOLEDB:*** Describes how to use UniOLEDB, an interface to UniVerse and UniData systems for OLE DB consumers. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with OLE DB, and who want to write OLE DB programs that access these databases.

# Introduction

About InterCall . . . . .	1-2
Minimum System Requirements . . . . .	1-3
InterCall Installation. . . . .	1-4
On Windows Platforms . . . . .	1-4
On UNIX Systems . . . . .	1-5
How InterCall Works . . . . .	1-6
Copying the Software . . . . .	1-7
UniVerse NLS in Client Programs . . . . .	1-8
NLS Configurable Parameters . . . . .	1-8
Character Mapping. . . . .	1-8
The Sample Program . . . . .	1-10

---

## About InterCall

InterCall is an API (application programming interface) that enables a UNIX or Windows client to access data on UniVerse and UniData servers. With InterCall, your applications can:

- Connect to one or more servers
- Access files and records
- Execute database commands and UniVerse BASIC programs

On Windows platforms, you can write applications for client programs using any development tool that accesses DLLs, for example, Visual Basic, C, or Visual C/C++. On UNIX, you can use any tool that accesses static libraries, typically a C compiler.

---

## Minimum System Requirements

To run InterCall applications, you need the following:

- On a UNIX server:
  - UniVerse Release 8.3.3.1G or later, or UniData Release 5.1 or later
  - TCP/IP
  - UniRPC daemon (*unirpcd*) running
- On a Windows server:
  - UniVerse Release 9.3.1 or later, or UniData Release 5.1 or later
  - TCP/IP, if connected to a UNIX client
  - TCP/IP or LAN Manager, if connected to a Windows client
  - UniRPC service (*unirpc*) running
- On a UNIX client:
  - TCP/IP
- On a Windows client:
  - TCP/IP, if connected to a UNIX server
  - TCP/IP or LAN Manager, if connected to a Windows server
  - Required InterCall files copied from the development system

To develop InterCall applications, C-language development tools must be available on the system you are using for development.



---

## InterCall Installation

The installation of InterCall is different on Windows platforms and UNIX platforms.

### On Windows Platforms

InterCall is one of several APIs in the UniDK (Uni Development Kit). The UniDK is installed using the standard Microsoft Windows installation procedure. The following UniDK files are used for InterCall development.

---

File	Description
include/intcall.h	A C header file used when compiling InterCall application programs.
include/INTCALL.TXT	A Visual Basic include file used when compiling InterCall application programs.
lib/UVIC32.LIB	A library file used when linking InterCall application programs.
bin/UVIC32.DLL	A DLL used by InterCall applications at run time.
bin/UVCLNT32.DLL	A DLL used by InterCall applications at run time.
bin/unirpc32.dll	A DLL used by InterCall applications at run time.

---

#### UniDK Files for InterCall Development

## On UNIX Systems

The InterCall SDK is included on the UniVerse and UniData installation tapes for UNIX systems. It is installed from the IC group as part of the standard installation. The installation process creates a directory called *icsdk* in the *unishared* directory, whose path is stored in the file */.unishared*. The *icsdk* directory contains the following files:

File	Description
<i>version</i>	A text file containing the version number of InterCall.
<i>intcall.h</i>	A header file used when compiling InterCall application programs.
<i>libuvic.a</i>	An archive file used when linking InterCall application programs.
<i>ictest.c</i>	C source code for a sample InterCall application.
<i>ictest.mak</i>	A make file for the sample InterCall application.

***icsdk* Files**

---

## How InterCall Works

The client program initializes InterCall by calling one of the following functions:

- **ic\_opensession**
- **ic\_unidata\_session**
- **ic\_universe\_session**

These functions log on to a host system over TCP/IP and run a server.

As the client program runs, InterCall functions send requests to the server to be executed. The program can access files, records, commands, and UniVerse BASIC programs that are available in the server database environment. When it finishes, it calls **ic\_quit** or **ic\_quitall** and the server program terminates.

---

## Copying the Software

You can make as many copies as you want of the client part of InterCall without further payment.

You may *not* copy or redistribute the server program unless your license agreement explicitly allows you to.

If you want to distribute part of InterCall with the executable version of an application that uses it, you can. The following files are required for 32-bit Windows systems:

- UVIC32.DLL
- UVCLNT32.DLL
- unirpc32.dll

**Note:** *On UNIX systems you do not need to redistribute any InterCall components.*



---

## UniVerse NLS in Client Programs

To use NLS features in client programs connected to a UniVerse database, NLS mode must be enabled on the server. All servers honor the settings of the database configurable parameters and client requests for character mapping and locales. For detailed information about NLS, see the *UniVerse NLS Guide*.

### NLS Configurable Parameters

You can configure the following parameters:

Parameter	Description
NLSDEFSRVLC	Specifies the name of the default locale to use for passing data to and from client programs. This locale is used if the client program does not specify a server locale. The default value is ISO8859-1+MARKS.
NLSDEFSRVMAP	Specifies the name of the default map to use for passing data to or from client programs. This map is used if the client program does not specify a server map. The default value is ISO8859-1+MARKS.
NLSLCMODE	Specifies whether locales are enabled. A value of 1 indicates that locales are enabled, a value of 0 indicates that locales are disabled. The default setting is 0. This parameter has no effect unless NLSMODE is set to 1.
NLSMODE	Turns NLS mode on or off. A value of 1 indicates that NLS is on, a value of 0 indicates NLS is off. If NLS mode is off, UniVerse does not check any other NLS parameters.

---

#### Configurable Parameters

### Character Mapping

UniVerse performs character mapping on the server. Your program can inform the server of the appropriate map name or the character set you use to send and receive data. In theory, you can set and reset maps as many times as you want in a program. All users who log on in a client/server system have their own individual copies of the server program.

## **Locale Conversion**

UniVerse does locale conversions on the server. Your program must inform the server of the locale your programs want to use in order to send or receive data. Once a connection to the server is established, you can change the locale settings as you wish. These settings do not interfere with other client/server users.

## **System Delimiters and the Null Value**

Your program must use the correct values for system delimiters and the null value. You should not use hard-coded values for system delimiters.

---

## The Sample Program

On UNIX systems, InterCall comes with a sample program called *ictest*, which prompts for a machine and account to connect to and then reads the ED record from that account's VOC. To use *ictest*, you must compile it by running the following command in the `icsdk` directory:

```
$ make -f ictest.mak
```

You may need to copy the directory, change its permissions, or both, before you can write to it.

**Note:** *ictest* is not available on Windows platforms.



---

# Programming with InterCall

Server Sessions . . . . .	2-3
Using the Microsoft Security Token . . . . .	2-4
Argument Passing Conventions . . . . .	2-5
The ICSTRING Type . . . . .	2-5
Calling Functions from C Programs . . . . .	2-5
Calling Functions from Visual Basic Programs . . . . .	2-6
Using the @TTY Variable . . . . .	2-7



This chapter describes programming with InterCall.

---

## Server Sessions

An InterCall client can support up to 10 simultaneous sessions on different database servers.

Open a session on a database server using one of the following functions:

- `ic_opensession`
- `ic_unidata_session`
- `ic_universe_session`

These functions return a unique session identifier for the new session. Each time you use one of these functions to open a new session on the server, the new session becomes the current session on which subsequent InterCall functions act.

Move among sessions using the `ic_setsession` function. **ic\_setsession** finds the desired session using the session identifier returned by **ic\_opensession**, **ic\_unidata\_session**, or **ic\_universe\_session**.

Get information about the current session using the `ic_session_info` function.

Close a session using the `ic_quit` function. **ic\_quit** closes any open files and releases locks in the current session, and closes the connection with the server. However, a closed session still will be the current session until you use **ic\_setsession** to switch to another session.

Use `ic_quitall` to close all open sessions.

When you open a file on the server using `ic_open`, InterCall returns a unique file identifier for that file within the current session. If you try to use this unique file identifier to reference the file from any other session, InterCall returns the error IE\_FIFS (file invalid for session).

### *Device Licensing*

Device licensing restricts InterCall connections, just as it restricts any other type of connection. Without device licensing, the connection limit is the lesser of the server's license limit, or 1024.

## Using the Microsoft Security Token

When you log on to a Windows client, your login details are held in a Microsoft Security Token.

You can use the security token to make a LAN pipes connection from InterCall to a Windows server.

To use the security token, set the *user\_name* and *password* parameters in the **ic\_opensession**, **ic\_unidata\_session**, or **ic\_universe\_session** expression to empty strings.

---

## Argument Passing Conventions

The functions exported by the InterCall library use only two types of argument: numeric arguments and string arguments. For more information about passing arguments, see the next four sections.

### The ICSTRING Type

ICSTRING is C structure. It is used by [ic\\_subcall](#) to pass arguments to the UniVerse BASIC cataloged subroutines.

The structure has the following definition in the *intcall.h* header file:

```
typedef struct icstring
{
    long len;
    unsigned char * text;
} ICSTRING;
```

*len* is used to store the length of the data being used. *text* is an unsigned char pointer that points to an area of memory containing the data.

Allocate the memory to which *text* points using [ic\\_malloc](#) or [ic\\_calloc](#). Release memory with [ic\\_free](#).

For more information about subroutines, see *UniVerse BASIC*.

### Calling Functions from C Programs

To use InterCall functions from an application written in C, include the header file *intcall.h*. This file contains ANSI-C function declarations for all InterCall functions. Here is an example:

```
void
ic_open (LPLONG , LPLONG , LPSTR , LPLONG , LPLONG , LPLONG
);
```

## Calling Functions from Visual Basic Programs

To use InterCall functions in a Visual Basic application, you should add the INTCALL.TXT file to your project. This file includes the necessary function declarations for all InterCall functions.

Create a new module and load INTCALL.TXT into it using the **Load Text** option under the File menu. This module can be added to any new program that needs to use InterCall procedures. For example:

```
Declare Sub ic_open Lib "UVIC.DLL" (FileID As Long, DictFlag As Long, ByVal FileName As String, FileLength As Long, StatusFunc As Long, Status As Long)
```

Numeric arguments are declared using `As Long`; character arguments are declared using `ByVal` and `As String`.

For information about string arguments used as output arguments, see Size of Output Buffers in the next section.

### *Size of Output Buffers*

It is important to make sure that any string variable used as an output argument to an InterCall function is large enough to hold the returned data. The size of the buffer passed to the InterCall function is the actual size of the value contained in the Visual Basic variable before the call.

One way to call an InterCall function that returns a string value is to make the returned argument long enough by filling it with characters:

```
Dim ListBuffer As String
ListBuffer = String$(1000, 0)
ic_readlist 0, ListBuffer, Len(ListBuffer), ListLength, ListCount,
ErrorCode
```

Another solution is to define the string as fixed length:

```
Dim ListBuffer As String * 1000
ic_readlist 0, ListBuffer, Len(ListBuffer), ListLength, ListCount,
ErrorCode
```

In either case, you should refer to the data returned by `ic_readlist` as:

```
Left$(ListBuffer, ListLength)
```

---

## Using the @TTY Variable

You can use the @TTY variable setting to determine the type of connection made to the server. The three possible settings are:

- The terminal number. This is the UNIX path of the terminal connected, or a number representing the connection from a Windows system.
- The string `phantom`. This is set if a phantom process is connected to the server.
- The string `uvcs` (on UniVerse systems) or `udcs` (on UniData systems). This is set if the connection is made from InterCall.

You can use this returned value, by adding a paragraph entry to the VOC file. For example:

```
PA
IF @TTY = 'uvcs' THEN GO END:
START.APP
END:
```

For more information, see the following functions in Chapter 3, “[InterCall Functions](#)”:

- [ic\\_execute](#)
- [ic\\_opensession](#)
- [ic\\_subcall](#)

# InterCall Functions

Function Summary . . . . .	3-5
Error Codes . . . . .	3-8
ic_alpha . . . . .	3-9
ic_calloc . . . . .	3-10
ic_cleardata . . . . .	3-11
ic_clearfile. . . . .	3-12
ic_clearselect . . . . .	3-14
ic_close. . . . .	3-16
ic_closeseq . . . . .	3-18
ic_data . . . . .	3-20
ic_date . . . . .	3-22
ic_delete . . . . .	3-23
ic_execute . . . . .	3-25
ic_executecontinue . . . . .	3-27
ic_extract . . . . .	3-29
ic_fileinfo . . . . .	3-31
ic_filelock . . . . .	3-34
ic_fileunlock . . . . .	3-36
ic_fmt . . . . .	3-38
ic_formlist. . . . .	3-40
ic_free . . . . .	3-42
ic_getlist . . . . .	3-43
ic_get_locale . . . . .	3-45
ic_get_map . . . . .	3-47
ic_get_mark_value . . . . .	3-49
ic_getvalue . . . . .	3-51
ic_iconv . . . . .	3-54

ic_indices . . . . .	3-56
ic_inputreply . . . . .	3-59
ic_insert. . . . .	3-61
ic_itype . . . . .	3-63
ic_locate . . . . .	3-65
ic_lock . . . . .	3-67
ic_lower . . . . .	3-68
ic_malloc . . . . .	3-70
ic_oconv . . . . .	3-71
ic_open . . . . .	3-73
ic_openseq . . . . .	3-76
ic_opensession . . . . .	3-78
ic_quit . . . . .	3-81
ic_quitall . . . . .	3-82
ic_raise . . . . .	3-83
ic_read . . . . .	3-85
ic_readblk . . . . .	3-88
ic_readlist . . . . .	3-90
ic_readnext. . . . .	3-92
ic_readseq . . . . .	3-94
ic_readv. . . . .	3-96
ic_recordlock . . . . .	3-99
ic_recordlocked . . . . .	3-102
ic_release . . . . .	3-105
ic_remove . . . . .	3-107
ic_replace . . . . .	3-110
ic_seek . . . . .	3-113
ic_select . . . . .	3-115
ic_selectindex . . . . .	3-117
ic_session_info . . . . .	3-119
ic_set_comms_timeout . . . . .	3-121
ic_set_locale . . . . .	3-123
ic_set_map (UniVerse only) . . . . .	3-125
ic_setsession . . . . .	3-127
ic_setvalue . . . . .	3-129
ic_strdel. . . . .	3-131



ic_subcall . . . . .	.3-133
ic_time . . . . .	.3-135
ic_timedate . . . . .	.3-136
ic_trans . . . . .	.3-138
ic_unlock . . . . .	.3-140
ic_unidata_session . . . . .	.3-141
ic_universe_session . . . . .	.3-144
ic_weofseq . . . . .	.3-147
ic_write . . . . .	.3-149
ic_writeblk . . . . .	.3-151
ic_writeseq . . . . .	.3-153
ic_writev . . . . .	.3-155

This chapter describes the InterCall functions in alphabetical order. All InterCall functions begin with the prefix **ic\_**. The syntax diagram for each function includes the function name and any applicable input and output variables. For example:

**ic\_alpha** (*string, string\_len, code*)

---

# Function Summary

The following table lists all InterCall functions. See also the general explanation of argument passing in [Using the Microsoft Security Token](#) and the description of the ICSTRING structure in [The ICSTRING Type](#) in Chapter 2, “[Programming with InterCall](#).”

Use	Function
Accessing a server	<a href="#">ic_opensession</a> <a href="#">ic_session_info</a> <a href="#">ic_setsession</a> <a href="#">ic_set_comms_timeout</a> <a href="#">ic_quit</a> <a href="#">ic_quitall</a> <a href="#">ic_unidata_session</a> <a href="#">ic_universe_session</a>
Reading and modifying records	<a href="#">ic_read</a> <a href="#">ic_readv</a> <a href="#">ic_writev</a> <a href="#">ic_write</a> <a href="#">ic_trans</a> <a href="#">ic_release</a> <a href="#">ic_delete</a>
Reading and modifying sequential files	<a href="#">ic_openseq</a> <a href="#">ic_readseq</a> <a href="#">ic_readblk</a> <a href="#">ic_writeseq</a> <a href="#">ic_writeblk</a> <a href="#">ic_weofseq</a> <a href="#">ic_seek</a> <a href="#">ic_closeseq</a>

---

Functions and Their Uses

Use	Function
Accessing and modifying strings	<code>ic_alpha</code> <code>ic_extract</code> <code>ic_fmt</code> <code>ic_iconv</code> <code>ic_insert</code> <code>ic_locate</code> <code>ic_lower</code> <code>ic_oconv</code> <code>ic_raise</code> <code>ic_remove</code> <code>ic_replace</code> <code>ic_strdel</code>
Accessing and modifying select lists	<code>ic_select</code> <code>ic_selectindex</code> <code>ic_getlist</code> <code>ic_readlist</code> <code>ic_formlist</code> <code>ic_readnext</code> <code>ic_clearselect</code>
Functions and Their Uses (Continued)	

Use	Function
Managing database files	<b>ic_open</b> <b>ic_fileinfo</b> <b>ic_filelock</b> <b>ic_fileunlock</b> <b>ic_recordlock</b> <b>ic_recordlocked</b> <b>ic_close</b> <b>ic_clearfile</b>
Using NLS (UniVerse only)	<b>ic_get_locale</b> <b>ic_get_map</b> <b>ic_get_mark_value</b> <b>ic_set_locale</b> <b>ic_set_map (UniVerse only)</b>
Using system utilities	<b>ic_calloc</b> <b>ic_cleardata</b> <b>ic_data</b> <b>ic_date</b> <b>ic_execute</b> <b>ic_executecontinue</b> <b>ic_free</b> <b>ic_getvalue</b> <b>ic_indices</b> <b>ic_inputreply</b> <b>ic_itype</b> <b>ic_lock</b> <b>ic_malloc</b> <b>ic_setvalue</b> <b>ic_subcall</b> <b>ic_time</b> <b>ic_timedate</b> <b>ic_unlock</b>

---

**Functions and Their Uses (Continued)**

---

## Error Codes

InterCall functions return error information as a status code. Symbolic constants representing each error number can be found in the files *intcall.h* (for C) and INTCALL.TXT (for Visual Basic) in the include subdirectory of the InterCall install directory. For a list of InterCall error codes, see Appendix B, “[Error Codes](#).”

---

# ic\_alpha

## Syntax

**ic\_alpha** (*string*, *string\_len*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>string</i>	char *	String to be tested.
<i>string_len</i>	long *	Length of string.

**ic\_alpha Input Variables**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_alpha Output Variable**

## Description

**ic\_alpha** determines whether a string is alphabetic or nonalphabetic. If the string contains the characters a through z or A through Z, it returns a value of 1. If the string contains any other characters or an empty string, it returns 0.

---

# ic\_calloc

## Syntax

*ptr* = **ic\_calloc** (*size*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>size</i>	long *	Number of bytes to allocate and clear.

**ic\_calloc Input Variable**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>ptr</i>	void *	Pointer to the allocated memory.

**ic\_calloc Output Variable**

## Description

**ic\_calloc** allocates and zeros a piece of memory for use by [ic\\_subcall](#), returning a pointer, *i*, to the allocated memory. Use *ptr* with [ic\\_free](#) to release the memory when it is no longer needed.

See also [The ICSTRING Type](#) in Chapter 2, “Programming with InterCall.”

## Related Function

[ic\\_malloc](#)



---

# ic\_cleardata

## Syntax

ic\_cleardata (*code*)

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	1 if the string was alphabetic, 0 if it was nonalphabetic, or a specific error code if execution was not successful.

ic\_cleardata Output Variable

## Description

ic\_cleardata flushes data loaded by the [ic\\_data](#) function from the input stack.

---

## ic\_clearfile

### Syntax

**ic\_clearfile** (*file\_id*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.

#### ic\_clearfile Input Variable

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_clearfile Output Variable

### Description

**ic\_clearfile** deletes all open data or dictionary records of a server database file.

**ic\_clearfile** deletes the record content only, not the file itself. You must specify each file to be cleared in a separate **ic\_clearfile** statement.

### Related Functions

[ic\\_close](#)

[ic\\_delete](#)

# ic\_clearselect

## Syntax

ic\_clearselect (*select\_list\_num*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>select_list_num</i>	long *	Identifies the select list (0 through 10) that is to be cleared.

ic\_clearselect Input Variable

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_clearselect Output Variable

## Description

ic\_clearselect clears an active select list. You can get an active select list by:

- Using [ic\\_select](#) to create it
- Using [ic\\_getlist](#) to restore it
- Using [ic\\_execute](#) to execute a database command such as SELECT, which creates a select list

## Related Functions

[ic\\_formlist](#)  
[ic\\_readlist](#)  
[ic\\_readnext](#)  
[ic\\_selectindex](#)

---

# ic\_close

## Syntax

`ic_close` (*file\_id*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.

**ic\_close Input Variable**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_close Output Variable**

## Description

`ic_close` closes an open server database file. Use `ic_close` after opening and processing a file. This function releases any file locks or record locks for the current user.

## Related Functions

[ic\\_filelock](#)  
[ic\\_fileunlock](#)  
[ic\\_recordlock](#)

# ic\_closeseq

## Syntax

ic\_closeseq (*file\_id*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.

ic\_closeseq Input Variable

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_closeseq Output Variable

## Description

*Note:* UniData databases do not support the **ic\_closeseq** function.

**ic\_closeseq** closes a file that was opened for sequential processing.



## Related Functions

[ic\\_readseq](#)  
[ic\\_weofseq](#)  
[ic\\_writeseq](#)



---

## ic\_data

### Syntax

**ic\_data** (*string*, *string\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>string</i>	char *	Data value to be placed in the queue.
<i>string_len</i>	long *	Length of the data value in bytes.

**ic\_data Input Variables**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_data Output Variable**

### Description

**ic\_data** loads strings onto the input stack. These values can be used as responses to UniVerse BASIC or UniBasic INPUT statements executed on the server.

Expressions used in **ic\_data** can be numeric or string data. **ic\_data** handles expressions in a first-in, first-out order.

**ic\_data** functionally is the same as the UniVerse BASIC or UniBasic DATA statement.

## Related Functions

[ic\\_cleardata](#)  
[ic\\_execute](#)  
[ic\\_executecontinue](#)

---

# ic\_date

## Syntax

ic\_date (*date*, *code*)

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>date</i>	long *	Server system date in internal format.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_data Output Variables

## Description

**ic\_date** returns the server system date in internal format. The internal format for the date is based on a reference date of December 31, 1967, which is day 0. All dates therefore are positive numbers representing the number of days elapsed since day 0.

## Related Functions

[ic\\_time](#)  
[ic\\_timedate](#)

# ic\_delete

## Syntax

**ic\_delete** (*file\_id*, *lock*, *record\_id*, *id\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>lock</i>	long *	Specifies what actions to perform if the user has locked the record: <div><div>IK_DELETE</div><div>Releases any locks you hold on that record. If another user has an exclusive lock on the record to be deleted, deletion fails and a locked error is returned.</div><div>IK_DELETEW</div><div>Pauses until the lock is released if another user has an exclusive update on the record to be deleted.</div><div>IK_DELETEU</div><div>Retains any locks held by the user on the record after deletion.</div></div>
<i>record_id</i>	char *	Character string containing the record ID of the record to be deleted.
<i>id_len</i>	long *	Actual length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.

**ic\_delete Input Variables**

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_delete</b> is executed.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_delete Output Variables

# Description

**ic\_delete** deletes a record from an open server database file. The value of *lock* specifies what actions **ic\_delete** performs. If the record is not found, *code* will contain the error IE\_RNF (record not found).

# Related Functions

- [ic\\_clearfile](#)
- [ic\\_close](#)
- [ic\\_release](#)

---

## ic\_execute

### Syntax

**ic\_execute** (*command*, *command\_len*, *text\_buffer*, *text\_buf\_size*, *text\_len*, *return\_code*, *return\_code\_2*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>command</i>	char *	Command to be executed.
<i>command_len</i>	long *	Length of the command in bytes.
<i>text_buffer</i>	long *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
<i>text_buf_size</i>	long *	Maximum size of the returned output in bytes.

---

#### ic\_execute Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_len</i>	long *	Actual length of the returned output in bytes.
<i>return_code</i>	long *	Value of the @SYSTEM.RETURN.CODE variable following execution of the command.
<i>return_code_2</i>	long *	The second part of the value of the @SYSTEM.RETURN.CODE variable following execution of the command. Following some database commands, @SYSTEM.RETURN.CODE contains two values, which are assigned to <i>return_code</i> and <i>return_code_2</i> . When @SYSTEM.RETURN.CODE has only one value, <i>return_code_2</i> is set to 0, and the value of @SYSTEM.RETURN.CODE is returned in <i>return_code</i> .
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_execute Output Variables

## Description

**ic\_execute** executes a server database command and copies the resulting output into a buffer supplied by the user. The output is truncated if it is larger than *text\_buf\_size* bytes in length and the error IE\_BTS (buffer too small) is returned. The output from a Windows server is always paged, whereas the output from a UNIX server is not. If the executed command requests input, then the function returns the output and the error code IE\_AT\_INPUT (server at input).

Use [ic\\_executecontinue](#) to supply an additional buffer to continue reading output when **ic\_execute** returns IE\_BTS.

See also [Using the @TTY Variable](#) in Chapter 2, “Programming with InterCall.”

## Related Function

[ic\\_inputreply](#)

---

## ic\_executecontinue

### Syntax

**ic\_executecontinue** (*text\_buffer*, *text\_buf\_size*, *text\_len*, *return\_code*, *return\_code\_2*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>text_buf_size</i>	long *	Maximum size of the returned output in bytes.

#### ic\_executecontinue Input Variable

### Output Variables

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
<i>text_len</i>	long *	Actual length of the returned output in bytes.
<i>return_code</i>	long *	The value of the @SYSTEM.RETURN.CODE variable following execution of the command.
<i>return_code_2</i>	long *	The second part of the @SYSTEM.RETURN.CODE variable following execution of the command. If a second part does not exist for the executed command, <i>return_code_2</i> is 0.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_executecontinue Output Variables



## Description

**ic\_executecontinue** lets you supply an additional buffer to continue reading output when **ic\_execute** returns IE\_BTS (buffer too small) in its *code* argument. If **ic\_executecontinue** returns IE\_BTS, use this function for as many times as necessary. The output from a Windows server is always paged, whereas the output from a UNIX server is not.

If **ic\_executecontinue** returns IE\_AT\_INPUT (server at input), use **ic\_inutreply**.

---

## ic\_extract

### Syntax

**ic\_extract** (*dynamic\_array*, *length\_da*, *field*, *value*, *subvalue*, *string*, *max\_str\_size*, *string\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	String containing a server database dynamic array.
<i>length_da</i>	long *	Length of the dynamic array in bytes.
<i>field</i>	long *	Number of the field to extract.
<i>value</i>	long *	Number of the value to extract.
<i>subvalue</i>	long *	Number of the subvalue to extract.
<i>max_str_size</i>	long *	Maximum size of the string buffer in bytes.

---

#### ic\_extract Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>string</i>	char *	Substring extracted from the dynamic array.
<i>string_len</i>	long *	Actual length of the extracted string in bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_extract Output Variables

## Description

**ic\_extract** returns data from a single field, value, or subvalue of a dynamic array. The numeric values of *field*, *value*, and *subvalue* determine which data is returned:

- If both *value* and *subvalue* are 0, the entire field is extracted.
- If only *subvalue* is 0, the entire value is extracted.
- If no argument is 0, the subvalue is extracted.

If the string buffer is too small for the specified field, value, or subvalue, IE\_BTS (buffer too small) is returned in *code*.

## Related Functions

[ic\\_insert](#)  
[ic\\_remove](#)  
[ic\\_replace](#)  
[ic\\_strdel](#)

---

## ic\_fileinfo

### Syntax

**ic\_fileinfo** (*key*, *file\_id*, *data*, *buffer*, *buffer\_size*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Specifies what information is required. The symbolic names for <i>key</i> are included in the <code>intcall.h</code> and <code>INTCALL.TXT</code> files.
<i>file_id</i>	long *	File identifier returned by a previous call to the <a href="#">ic_open</a> function.
<i>buffer_size</i>	long *	Length of <i>buffer</i> in bytes.

---

#### ic\_fileinfo Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>data</i>	long *	Information sought or its length.
<i>buffer</i>	char *	Buffer where returned string information will be placed.
<i>code</i>	long *	Either 0 or <code>IE_STR</code> if execution was successful or a specific error code if execution was not successful.

---

#### ic\_fileinfo Output Variables

## Description

**ic\_fileinfo** returns a specified item of information relating to an open server database file. For multivolume or distributed files, most data items are returned as a dynamic array, one element for each part.

If the returned data is numeric, its value is placed in *data*, and *code* is set to 0. The *buffer* argument is not used. If the data item is a string or a dynamic array, it is returned in *buffer*, *code* is set to IE\_STR, and *data* is set to the length of data returned, in bytes.

If the size of the returned output exceeds *buffer\_size* bytes, IE\_BTS (buffer too small) is returned in *code*. Any other value is a specific error code.

The following table lists the valid values for *key*.

Value	Symbolic Name	Description
0	FINFO_IS_FILEVAR	1 if <i>file_id</i> is a valid file identifier; 0 otherwise.
1	FINFO_VOCNAME	VOC name of file.
2	FINFO_PATHNAME	Path of the file.
3	FINFO_TYPE	File type as follows: 2 = UniData static hashed file 3 = Dynamic 4 = Type 1 5 = Sequential 7= Distributed and Multivolume
4	FINFO_HASHALG	Hashing algorithm as follows: 0 = UniData algorithm 0 1 = UniData algorithm 1 2 = GENERAL 3 = SEQ.NUM
5	FINFO_MODULUS	Current modulus.
6	FINFO_MINMODULUS	Minimum modulus.
7	FINFO_GROUPSIZE	Group size, in 1Kbyte units.
8	FINFO_LARGERRECORDSIZE	Large record size.

### key Values

Value	Symbolic Name	Description
9	FINFO_MERGELOAD	Merge load parameter.
10	FINFO_SPLITLOAD	Split load parameter.
11	FINFO_CURRENTLOAD	Current loading of the file (%).
12	FINFO_NODENAME	Empty string, if the file resides on the local system, otherwise the name of the node where the file resides.
13	FINFO_IS_AKFILE	1 if secondary indexes exist on the file; 0 otherwise.
14	FINFO_CURRENTLINE	Current line number.
15	FINFO_PARTNUM	For a distributed file, returns list of currently open part numbers.
16	FINFO_STATUS	For a distributed file, returns list of status codes showing whether the last I/O operation succeeded or failed for each part. A value of -1 indicates the corresponding part file is not open.
19	FINFO_IS_FIXED_MODULUS	1 if the file has a fixed modulus; 0 otherwise.

#### *key Values*

## Related Function

[ic\\_extract](#)

# ic\_filelock

## Syntax

**ic\_filelock** (*file\_id*, *status\_func*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.

ic\_filelock Input Variable

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_filelock</b> is executed:
		0                    There is a IK_READU lock on the file.
		other                Terminal number of the user who has set an exclusive lock on the file.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_filelock Output Variables

## Description

**ic\_filelock** locks an open server database file, preventing any other user from:

- Reading any record in that file using the [ic\\_read](#) function with a IK\_READU or IK\_READL lock, or any of the UniVerse BASIC or UniBasic READL, READU, READVU, or MATREADU statements
- Reading any field in that file using the [ic\\_readv](#) function with a IK\_READU lock or the UniVerse BASIC or UniBasic READVU statement
- Locking the file using **ic\_filelock** or the UniVerse BASIC or UniBasic FILELOCK statement

Use [ic\\_fileunlock](#) (or [ic\\_release](#)) to unlock files after using them.

## Related Functions

[ic\\_close](#)  
[ic\\_write](#)  
[ic\\_writev](#)



# ic\_fileunlock

## Syntax

ic\_fileunlock (*file\_id*, *status\_func*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.

ic\_fileunlock Input Variable

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_unfilelock</b> is executed:  -2      The file was not locked before execution.  0      The file was locked before executing <b>ic_fileunlock</b> . <i>status_func</i> is always 0 for database files.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_fileunlock Output Variables

## Description

**ic\_fileunlock** unlocks a file locked by [ic\\_filelock](#). **ic\_fileunlock** does not unlock records that were locked using:

- [ic\\_read](#) with a IK\_READL or IK\_READU lock
- [ic\\_readv](#) with a IK\_READU lock
- The UniVerse BASIC or UniBasic statements READL, READU, READVU, or MATREADU

## Related Functions

[ic\\_close](#)

[ic\\_write](#)

[ic\\_writev](#)

---

## ic\_fmt

### Syntax

**ic\_fmt** (*format, format\_len, string, string\_len, result, max\_rslt\_size, result\_len, status\_func*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>format</i>	char *	Format pattern to use. See “Description.”
<i>format_len</i>	long *	Length of the format pattern in bytes.
<i>string</i>	char *	Input string to be formatted.
<i>string_len</i>	long *	Length of <i>string</i> in bytes.
<i>max_rslt_size</i>	long *	Maximum size of the result buffer in bytes.

---

#### ic\_fmt Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>result</i>	char *	Buffer containing the formatted string.
<i>result_len</i>	long *	Length of the formatted string in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_fmt</b> is executed:

---

#### ic\_fmt Output Variables

Parameter	Type	Description
	0	The conversion was successful.
	1	The string expression passed as an argument is invalid.
	2	The conversion code passed as an argument to the function is invalid.

**ic\_fmt Output Variables (Continued)**

**Description**

**ic\_fmt** formats a string into various patterns. You can specify the following characteristics:

- Width of the field
- Fill character
- Right-justification or left-justification
- Numeric conversion specification
- Masking

**Related Functions**

[ic\\_iconv](#)  
[ic\\_oconv](#)

---

# ic\_formlist

## Syntax

**ic\_formlist** (*dynamic\_array*, *dynamic\_len*, *selnum*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	Dynamic array to load as a select list.
<i>dynamic_len</i>	long *	Length of the dynamic array.
<i>selnum</i>	long *	Select list number into which to load the dynamic array.

**ic\_formlist Input Variables**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_formlist Output Variable**

## Description

**ic\_formlist** produces a select list from a dynamic array.

## Related Functions

[ic\\_clearselect](#)  
[ic\\_getlist](#)  
[ic\\_readlist](#)  
[ic\\_readnext](#)  
[ic\\_select](#)

---

# ic\_free

## Syntax

`ic_free` (*ptr*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>ptr</i>	void *	Pointer to memory to be released.

### ic\_free Input Variable

Parameter	Type	Description
<i>ptr</i>	void *	Pointer to memory to be released.

## Description

`ic_free` releases a piece of memory previously allocated by [ic\\_malloc](#) or [ic\\_calloc](#).

See also [The ICSTRING Type](#) in Chapter 2, “[Programming with InterCall](#).”

## Related Function

[ic\\_subcall](#)

---

# ic\_getlist

## Syntax

**ic\_getlist** (*list\_name*, *length*, *select\_list\_num*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>list_name</i>	char *	Name of the select list to be restored.
<i>length</i>	long *	Length of <i>list_name</i> in bytes.
<i>select_list_num</i>	long *	Select list number (0 through 10) to be assigned to the restored select list.

---

### ic\_getlist Input Variables

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

### ic\_getlit Output Variable

## Description

**ic\_getlist** restores a select list from the &SAVEDLISTS& file. This function allows an application to use the record IDs previously saved by the SAVE.LIST command.



## Related Functions

[ic\\_clearselect](#)  
[ic\\_formlist](#)  
[ic\\_readlist](#)  
[ic\\_readnext](#)  
[ic\\_select](#)

---

## ic\_get\_locale

### Syntax

**ic\_get\_locale** (*key*, *locale\_string*, *max\_buff\_size*, *locale\_string\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Specifies the locale information you want to retrieve. It must be one of the following tokens: IK_LC_ALL = All categories IK_LC_TIME = Time category IK_LC_NUMERIC = Numeric category IK_LC_MONETARY = Monetary category IK_LC_CTYPE = Ctype category IK_LC_COLLATE = Collate category
<i>max_buff_size</i>	long *	Maximum size of the return buffer.

#### ic\_get\_locale Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>locale_string</i>	char *	Locale string for the current server locale.
<i>locale_string_len</i>	long *	Length of the returned string.
<i>code</i>	long *	Either 0 if the operation is successful or a specific error code if the operation failed.

#### ic\_get\_locale Output Variables



## Description

***Note:** UniData databases do not support the **ic\_get\_locale** function.*

**ic\_get\_locale** retrieves the name of the locale that the server is using.

If NLS mode is not enabled on the server, the error code IE\_NO\_NLS is returned in *code*.

If you specify IK\_LC\_ALL, all five category settings are returned, separated by the current field mark character. (You must use [ic\\_get\\_mark\\_value](#) to determine the value of the field mark character.)

See also “[UniVerse NLS in Client Programs](#)” in Chapter 1, “[Introduction](#).”

---

## ic\_get\_map

### Syntax

**ic\_get\_map** (*map\_name*, *max\_buff\_size*, *map\_name\_len*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>max_buff_size</i>	long *	Maximum size of the return buffer.

**ic\_get\_map Input Variable**

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>map_name</i>	char *	Map name of the current server map.
<i>map_name_len</i>	long *	Length of the returned string.
<i>code</i>	long *	Either 0 if the operation is successful or a specific error code if the operation failed.

**ic\_get\_map Output Variables**

### Description

**Note:** UniData databases do not support the **ic\_get\_map** function.

**ic\_get\_map** retrieves the name of the map currently used on the server.



If NLS mode is not enabled on the server, the error code IE\_NO\_NLS is returned in *code*. For more information, see [“UniVerse NLS in Client Programs”](#) in Chapter 1, [“Introduction.”](#)

---

## ic\_get\_mark\_value

### Syntax

**ic\_get\_mark\_value** (*key*, *mark\_string*, *max\_buff\_size*, *mark\_string\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Type of system delimiter. It must be one the following tokens: IK_IM = Item mark IK_FM = Field mark IK_VM = Value mark IK_SM = Subvalue mark IK_TM = Text mark IK_NULL = The null value
<i>max_buff_size</i>	long *	Maximum size of the return buffer.

#### ic\_get\_mark\_value Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>mark_string</i>	char *	Character value of the system delimiter.
<i>mark_string_len</i>	long *	Length of the returned string.
<i>code</i>	long *	Either 0 if the operation is successful or a specific error code if the operation failed.

#### ic\_get\_mark\_value Output Variables



## Description

***Note:** UniData databases do not support the **ic\_get\_mark\_value** function.*

A system delimiter that is used in the current character set on the server.

The system delimiter values retrieved are valid only for the current connection with the server. If NLS mode is off, the default values for these tokens (128, 251 to 255) are returned. For more information, see “[UniVerse NLS in Client Programs](#)” in Chapter 1, “[Introduction.](#)”

---

# ic\_getvalue

## Syntax

**ic\_getvalue** (*key*, *text\_buffer*, *buffer\_size*, *text\_len*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	A key value indicating which system variable is required.
<i>buffer_size</i>	long *	Maximum size of <i>text_buffer</i> in bytes.

---

### ic\_getvalue Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer that contains the value returned, as a string of characters.
<i>text_len</i>	long *	Length of data actually placed in <i>text_buffer</i> in bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

### ic\_getvalue Output Variables

## Description

**ic\_getvalue** returns the value of a system variable from the server program.



The following table lists the valid values for *key*.

<b>Value</b>	<b>Symbolic Name</b>	<b>Equivalent to</b>	<b>Description</b>
1	IK_AT_LOGNAME	@LOGNAME	The user login name.
2	IK_AT_PATH	@PATH	Path of the current account.
3	IK_AT_USERNO	@USERNO	The user number on a Windows server, and the process ID on a UNIX server.
4	IK_AT_WHO	@WHO	The name of the current database account.
5	IK_AT_TRANSACTION	@TRANSACTION.ID	A numeric value. Any nonzero value indicates that a transaction is active; 0 indicates that no transaction exists.
6	IK_AT_DATA_PENDING	@DATA.PENDING	Dynamic array containing input generated by the DATA statement.
7	IK_AT_USER_RETURN_CODE	@USER.RETURN.CODE	Status codes created by the user.

***key Values***

Value	Symbolic Name	Equivalent to	Description
8	IK_AT_SYSTEM_RETURN_CODE	@SYSTEM.RETURN.CODE	Status codes returned by system processes.
9	IK_AT_NULL_STR	@NULL.STR	The internal representation of the null value, which is CHAR (128) on UniVerse systems and CHAR(129) on UniData systems.
10	IK_AT_SCHEMA	@SCHEMA	Schema name of the current database account.

*key Values (Continued)*

## Related Function

[ic\\_setvalue](#)

---

# ic\_iconv

## Syntax

**ic\_iconv** (*conv, conv\_len, string, string\_len, result, max\_rslt\_size, result\_len, status\_func*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>conv</i>	char *	Conversion code to be used.
<i>conv_len</i>	long *	Length of the conversion code in bytes.
<i>string</i>	char *	String to be converted.
<i>string_len</i>	long *	Length of the string to be converted in bytes.
<i>max_rslt_size</i>	long *	Maximum size of <i>result</i> in bytes.

**ic\_iconv Input Variables**

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>result</i>	char *	Buffer containing the converted string.
<i>result_len</i>	long *	Length of the converted string in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_iconv</b> is executed:
	0	The conversion was successful.

**ic\_iconv Output Variables**

Parameter	Type	Description
	1	<i>string</i> was invalid. An empty string is returned, unless <i>string</i> is the null value, in which case <i>string</i> is returned.
	2	<i>conv</i> was invalid.
	3	The conversion was successful; there could be an invalid date.
	other	Any other value is an error code.

#### ic\_iconv Output Variables (Continued)

## Description

**ic\_iconv** converts a character string to an internal format specified by the conversion code. This function also can be used to check the validity of data. It is equivalent to the UniVerse BASIC or UniBasic ICONV function.

If *status\_func* equals 1 or 2, the string returned by **ic\_iconv** is the input string.

## Related Functions

[ic\\_alpha](#)  
[ic\\_fmt](#)  
[ic\\_oconv](#)

---

# ic\_indices

## Syntax

**ic\_indices** (*file\_id*, *ak\_name*, *ak\_name\_len*, *text\_buffer*, *text\_buf\_size*, *text\_len*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by a previous call to <a href="#">ic_open</a> .
<i>ak_name</i>	char *	Name of the secondary index on the file.
<i>ak_name_len</i>	long *	Length of <i>ak_name</i> . If <i>ak_name_len</i> is 0 or negative, a dynamic array is returned containing the secondary index names for all indexes on the file.
<i>text_buf_size</i>	long *	Maximum size of the returned output in bytes.

### ic\_indices Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
<i>text_len</i>	long *	Actual length of the returned output in bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

### ic\_indices Output Variables

## Description

**ic\_indices** returns information about the secondary indexes on a file. You can find the name of secondary indexes or return specific information about an index.

### *Finding the Name of Secondary Indexes*

If **ic\_indices** is called with a 0 or negative value for *ak\_name\_len*, it returns a dynamic array containing the secondary index names for all indexes in the file. If the file is of the wrong type, or if it has no indexes, a null string is returned. The index names are not in any particular order, and are separated by field marks. As the parts of a distributed file can be accessed individually, the list of secondary index names contains only the names of indexes that are common to all part files.

### *Returning Information About a Secondary Index*

If **ic\_indices** is called with a nonzero value for *ak\_name\_len*, it returns information about the index specified by *ak\_name*. If the named index does not exist, an empty string is returned. If the index exists, the form of the result depends on the type of index:

- If the named index is a D-type index, the result is a dynamic array containing the letter D in the first character of field 1 and the location number of the indexed field in field 2.
- If the named index is an I-type index, the result is a dynamic array containing the letter I in the first character of field 1, the I-type expression in field 2, and the compiled I-type code beginning in field 19. Fields 3 through 5 and 7 through 18 are empty.

For both D-type indexes and I-type indexes:

- If the index needs to be rebuilt, the second value of field 1 is 1, otherwise it is null.
- If the index is null-suppressed, the third value of field 1 is 1, otherwise it is null.
- If automatic updates are disabled, the fourth value of field 1 is 1, otherwise it is null.

Field 6 contains an S if the index is singlevalued, and an M if the index is multivalued.

## Related Function

[ic\\_selectindex](#)

---

# ic\_inputreply

## Syntax

**ic\_inputreply** (*reply\_string*, *reply\_len*, *add\_newline*, *text\_buffer*, *text\_buf\_size*, *text\_len*, *return\_code*, *return\_code\_2*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>reply_string</i>	char *	Characters that are to be supplied to the running program as terminal input.
<i>reply_len</i>	long *	Length of the reply string in bytes.
<i>add_newline</i>	long *	Indicates whether the input should be terminated with a return (NEWLINE) character. If <i>add_newline</i> is False (0), no newline is added; if <i>add_newline</i> is True (any nonzero value), the running program receives the characters of <i>reply_string</i> followed by a newline.
<i>text_buf_size</i>	long *	Maximum size of the returned output in bytes.

---

### ic\_inputreply Input Variables



# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
<i>text_len</i>	long *	Actual length of the returned output in bytes.
<i>return_code</i>	long *	Value of @SYSTEM.RETURN.CODE following the execution of this command.
<i>return_code_2</i>	long *	Second part of the @SYSTEM.RETURN.CODE variable following execution of the command. If a second part does not exist for the executed command, <i>return_code_2</i> is 0.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_inputreply Output Variables

## Description

**ic\_inputreply** responds to a request for input during execution of a command. **ic\_inputreply** lets you send data to a server at input (IE\_AT\_INPUT).

When **ic\_execute** returns IE\_AT\_INPUT in *code*, indicating that the command being executed is waiting for terminal input, use **ic\_inputreply** to supply characters as if they had come from a terminal. If **ic\_inputreply** also returns IE\_AT\_INPUT, you can call **ic\_inputreply** again, as many times as necessary.

If **ic\_inputreply** returns IE\_BTS (buffer too small), use **ic\_executecontinue**.

---

## ic\_insert

### Syntax

**ic\_insert** (*dynamic\_array*, *max\_da\_size*, *length\_da*, *field*, *value*, *subvalue*, *string*, *string\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>max_da_size</i>	long *	Maximum size of the <i>dynamic_array</i> buffer in bytes.
<i>field</i>	long *	Number of the field at which to insert the new string.
<i>value</i>	long *	Number of the value at which to insert the new string.
<i>subvalue</i>	long *	Number of the subvalue at which to insert the new string.
<i>string</i>	char *	String to insert into the dynamic array.
<i>string_len</i>	long *	Length of the new string in bytes.

**ic\_insert Input Variables**

### Input/Output Variables

The following table describes the input/output variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	String containing a server database dynamic array.
<i>length_da</i>	long *	Length of the old and new dynamic array in bytes.

**ic\_insert Input/Output Variables**

# Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_insert Output Variable

## Description

**ic\_insert** inserts a new field, value, or subvalue into a dynamic array at a specified location, returning the new dynamic array and its new length. The data content to be inserted is specified by *string*. The numeric values of *field*, *value*, and *subvalue* determine whether the new data is inserted as a field, value, or subvalue.

- If both *value* and *subvalue* are 0, the new data is inserted before the specified field.
- If only *subvalue* is 0, the new data is inserted before the specified value.
- If no argument is 0, the new data is inserted before the specified subvalue.

If the number of characters to be added extends the length of the dynamic array past *max\_da\_size*, the original dynamic array is not altered and an error value is returned in *code*.

## Related Functions

[ic\\_extract](#)  
[ic\\_remove](#)  
[ic\\_replace](#)  
[ic\\_strdel](#)

---

## ic\_itype

### Syntax

**ic\_itype** (*filename, filename\_len, record\_id, record\_id\_len, itype\_id, itype\_id\_len, text\_buffer, text\_buf\_size, text\_len, code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>filename</i>	char *	Name of the server database file.
<i>filename_len</i>	long *	Length of the file name in bytes.
<i>record_id</i>	char *	Record ID of the data record to be supplied as data during the evaluation.
<i>record_id_len</i>	long *	Length of the data record ID in bytes.
<i>itype_id</i>	char *	Record ID of the I-descriptor record which is to be evaluated.
<i>itype_id_len</i>	long *	Length of the I-descriptor record ID in bytes.
<i>text_buf_size</i>	long *	Size of <i>text_buffer</i> in bytes.

---

#### ic\_itype Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer that will contain the value returned, as a string of characters.
<i>text_len</i>	long *	Actual length of data placed in <i>text_buffer</i> in bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_itype Output Variables

# Description

**ic\_itype** evaluates an I-descriptor taken from the dictionary of a server database file, and returns the result. When **ic\_itype** is executed, the server attempts to open the dictionary of the specified file on the server, and then read the I-descriptor record.

If the I-descriptor is valid, it will always be evaluated. The system variable @ID is set to the value of *record\_id*, and the variable @RECORD is set to the contents of the specified record, and the result is placed in *text\_buffer*. If the data cannot be opened, if *record\_id* is null, or if the data record is not present, @RECORD is set to a null string instead.

If the dictionary cannot be opened, if the I-descriptor record is not present in the file, or if the I-descriptor field has not been compiled, then *code* will be set to an error code.

---

## ic\_locate

### Syntax

**ic\_locate** (*search, search\_len, dynamic\_array, dynamic\_len, field, value, start, order, order\_len, index, found, code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>search</i>	char *	Content of the field, value, or subvalue being sought.
<i>search_len</i>	long *	Length of the search string.
<i>dynamic_array</i>	char *	Dynamic array to be searched.
<i>dynamic_len</i>	long *	Length of the dynamic array.
<i>field</i>	long *	Starting field position for the search.
<i>value</i>	long *	Starting value position for the search.
<i>start</i>	long *	Field or value from which to start the search.
<i>order</i>	char *	String indicating the order of the elements within the dynamic array.
<i>order_len</i>	long *	Length of <i>order</i> is as follows: AL or A = Ascending, left-justified AR = Ascending, right-justified D = Descending, left-justified DR = Descending, right-justified
<b>ic_locate Input Variables</b>		

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>index</i>	long *	A variable to receive the position within the dynamic array of the string being sought.
<i>found</i>	long *	Either 1 if <i>search</i> was found or 0 if it was not.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_locate Output Variables

# Description

**ic\_locate** searches a dynamic array for a string and returns a value indicating whether the expression is in the array and where it is or where the expression should go if it is not in the array. **ic\_locate** searches the dynamic array for *search* and returns values indicating the following:

- Where *search* was found in the dynamic array
- Where *search* should be inserted in the dynamic array if it was not found

The search can start anywhere in *dynamic\_array.field* and *value* delimiter values specify:

- Where the search is to start in the dynamic array
- What kind of element is being searched for

# Related Functions

[ic\\_extract](#)  
[ic\\_insert](#)  
[ic\\_replace](#)  
[ic\\_strdel](#)

---

## ic\_lock

### Syntax

**ic\_lock** (*lock\_num*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>lock_num</i>	long *	An integer from 0 through 63 specifying one of the 64 public process locks.

**ic\_lock Input Variable**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_lock Output Variable**

### Description

**ic\_lock** sets a public process lock. The locks are used to protect user-defined resources or events on the server from unauthorized or simultaneous data file access by different users.

### Related Function

[ic\\_unlock](#)



---

# ic\_lower

## Syntax

**ic\_lower** (*string*, *string\_len*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>string_len</i>	long *	Length of the string used.

**ic\_lower Input Variable**

## Input/Output Variable

The following table describes the input/output variable.

Parameter	Type	Description
<i>string</i>	char *	String in which the delimiters are lowered.

**ic\_lower Input/Output Variable**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_lower Output Variable**

## Description

**ic\_lower** returns *string* with the system delimiters converted to the next lower-level delimiter. For example, field marks are changed to value marks, value marks are changed to subvalue marks, and so on.

## Related Function

[ic\\_raise](#)

---

# ic\_malloc

## Syntax

*ptr* = **ic\_malloc** (*size*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>size</i>	long *	Number of bytes to allocate.

**ic\_malloc Input Variable**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>ptr</i>	void *	Pointer to the allocated memory.

**ic\_malloc Output Variable**

## Description

**ic\_malloc** allocates a piece of memory for use by [ic\\_subcall](#), returning a pointer, *ptr*, to the allocated memory. Use *ptr* with [ic\\_free](#) to release the memory when it is no longer needed.

See also [The ICSTRING Type](#) in Chapter 2, “[Programming with InterCall](#).”

## Related Function

[ic\\_calloc](#)

---

## ic\_oconv

### Syntax

**ic\_oconv** (*conv, conv\_len, string, string\_len, result, max\_rslt\_size, result\_len, status\_func*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>conv</i>	char *	Conversion code to be used.
<i>conv_len</i>	long *	Length of the conversion code in bytes.
<i>string</i>	char *	String to be converted.
<i>string_len</i>	long *	Length of the string to be converted in bytes.
<i>max_rslt_size</i>	long *	Maximum size of <i>result</i> in bytes.

**ic\_oconv Input Variables**

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>result</i>	char *	Buffer containing the converted string.
<i>result_len</i>	long *	Length of the converted string in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_oconv</b> is executed:

**ic\_oconv Output Variables**

Parameter	Type	Description
	-1	The conversion was completed but precision was lost. This value only occurs when numeric conversions are specified.
	0	The conversion was successful.
	1	The input string was invalid or the returned string exceeded <i>max_rslt_size</i> .
	2	The conversion code was invalid.
	other	Any other value is an error code.

**ic\_oconv Output Variables (Continued)**

**Description**

**ic\_oconv** converts a character string to an external format specified by the conversion code. This function is equivalent to the UniVerse BASIC or UniBasic OCONV function. Any string returned by [ic\\_iconv](#) can be supplied directly to **ic\_oconv** using the same conversion code.

If *status\_func* equals 1 or 2, the string returned by **ic\_oconv** is the input string.

**Related Functions**

[ic\\_alpha](#)  
[ic\\_fmt](#)

---

## ic\_open

### Syntax

**ic\_open** (*file\_id*, *dict\_flag*, *filename*, *file\_len*, *status\_func*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier that should be used for all subsequent operations on this file.
<i>dict_flag</i>	long *	Indicates whether the data or dictionary file is to be opened.
<i>filename</i>	char *	VOC name of the file to be opened.
<i>file_len</i>	long *	Length of <i>filename</i> in bytes.

#### ic\_open Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_open</b> is executed.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_open Output Variables

## Description

**ic\_open** opens a server database file for reading, writing, and deleting. You must open a database file with **ic\_open** before any type of file I/O operation can be executed.

The value of *dict\_flag* determines whether a data or dictionary file is opened. To open a dictionary file, the value of *dict\_flag* must be IK\_DICT. To open a data file, its value must be IK\_DATA.

The **ic\_open** function returns an integer as the unique file identifier in *file\_id*. All subsequent file operations should use this argument when referring to the opened file. If **ic\_open** does not execute successfully, the value of *file\_id* is 0.

The **ic\_open** function uses the file's VOC record in the account specified in the [ic\\_opensession](#), [ic\\_unidata\\_session](#), or [ic\\_universe\\_session](#) call. The VOC file record must be a valid file definition record.

Use separate **ic\_open** functions for each file. Any number of files can be opened at any point in the program.

For UniVerse files, *status\_func* contains the external file type, 2 through 18, 25, or 30. For all other server database files, *status\_func* contains the internal file type: 3 for a dynamic, multivolume or distributed file, or 4 for a type 1 file. If the file is not opened, *status\_func* contains 0.

For UniData files, *status\_func* returns either success or failure.

The following table lists the valid UniVerse values for *status\_func*.

Value	Meaning
-1	File name not found in the VOC file.
-2*	No file name or file. This error may occur when you cannot open a file across UV/Net.
-3	UNIX access error that occurs when you do not have UNIX permissions to access a database file in a UNIX directory. For example, this may occur when trying to access a type 1 or type 30 file.
-4*	Access error when you do not have UNIX permissions or if DATA.30 is missing for a type 30 file.

UniVerse *status\_func* Values

Value	Meaning
–5	Read error detected by UNIX.
–6	Unable to lock file header.
–7	Invalid file revision or wrong byte-ordering for the platform.
–8*	Invalid part file information.
–9*	Invalid type 30 file information in a distributed file.
–10	A problem occurred while the file was being rolled forward during warmstart recovery. Therefore, the file is marked “inconsistent”.
–11	The file is a view, therefore it cannot be opened by a UniVerse BASIC or UniBasic program.
–12	No SQL privilege to open the table.
–13*	Index problem.
–14	Cannot open an NFS file.
–99	(UniData only) The <b>ic_open</b> function failed.

#### UniVerse *status\_func* Values (Continued)

\* A generic error that can occur for a variety of reasons.

## Related Functions

[ic\\_close](#)  
[ic\\_filelock](#)  
[ic\\_fileunlock](#)  
[ic\\_read](#)  
[ic\\_readv](#)  
[ic\\_recordlock](#)  
[ic\\_write](#)  
[ic\\_writev](#)



# ic\_openseq

## Syntax

**ic\_openseq** (*file\_id*, *filename*, *file\_len*, *record\_id*, *record\_id\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier that should be used for all subsequent operations on this file.
<i>filename</i>	char *	VOC name of the file to be opened.
<i>file_len</i>	long *	Length of <i>filename</i> in bytes.
<i>record_id</i>	char *	Record in the file to be opened.
<i>record_id_len</i>	long *	Length of <i>record_id</i> .

ic\_openseq Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC STATUS function after <b>ic_openseq</b> is executed:
	0	<i>record_id</i> could not be found.

ic\_openseq Output Variables



Parameter	Type	Description
		1 <i>filename</i> is not type 1 or type 19.
		2 <i>filename</i> could not be found.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_openseq Output Variables (Continued)**

**Description**

***Note:** UniData databases do not support the **ic\_openseq** function.*

**ic\_openseq** opens a file for sequential processing, such as by [ic\\_readseq](#) or [ic\\_writeseq](#). A database file must be opened using the **ic\_openseq** function before any type of sequential file processing operation can be performed.

UniData databases do not support the **ic\_openseq** function.

**Related Functions**

- [ic\\_closeseq](#)
- [ic\\_readblk](#)
- [ic\\_seek](#)
- [ic\\_writeblk](#)

# ic\_opensession

## Syntax

*session\_id* = **ic\_opensession** (*server\_name*, *user\_name*, *password*, *account*, *status*, *subkey*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>server_name</i>	char *	Name of the server to which to connect. See “Description” for details on how you can use this to specify the transport type to use for the connection.
<i>user_name</i>	char *	Name to use to log on to the server.
<i>password</i>	char *	Password for <i>user_name</i> .
<i>subkey</i>	char *	Name of the device subkey, used when an application connects to a database server through a multiple-tier connection.
<i>account</i>	char *	Name or path of the account to access on the server.

ic\_opensession Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>session_id</i>	long	Unique session identifier.
<i>status</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic\_opensession Output Variables



## Description

**ic\_opensession** opens a new session from the client to a UniVerse or UniData server, and returns *session\_id*, a unique session identifier. Use the [ic\\_setsession](#) function to switch among sessions using the contents of *session\_id*.

An InterCall client can support up to 10 simultaneous sessions to different database servers.

**Note:** **ic\_opensession** *does not execute the LOGIN entry.*

**ic\_opensession** always opens a UniVerse or UniData server called *defcs*, which is defined in the *unirpcservices* file. Use [ic\\_universe\\_session](#) to open the UniVerse server *uvcs*. Use [ic\\_unidata\\_session](#) to open the UniData server *udcs* or another UniData server.

The way in which you specify *server\_name* determines the transport type used for the connection:

- If you enter *server\_name* only, the connection is made using TCP/IP.
- If you enter *server\_name*:TCP, the connection is made using TCP/IP.
- If you enter *server\_name*:LAN, the connection is made using LAN pipes.

For TCP/IP connections, you can also specify the port number and/or the IP address to use as part of *server\_name*. For example:

- If you enter *IP address* only, a TCP/IP connection is made to the specified address.
- If you enter *server\_name:port\_number*, a TCP/IP connection is made to the specified port number on the server.
- If you enter *IP address:port\_number*, a TCP/IP connection is made to the specified port number at the given IP address.

See also the following sections in Chapter 2, “[Programming with InterCall](#).”

- [Server Sessions](#)
- [Using the @TTY Variable](#)
- [Using the Microsoft Security Token](#)

## Related Functions

[ic\\_quit](#)  
[ic\\_quitall](#)  
[ic\\_unidata\\_session](#)  
[ic\\_universe\\_session](#)

---

# ic\_quit

## Syntax

**ic\_quit** (*code*)

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_quit Output Variable**

## Description

**ic\_quit** terminates the current InterCall session, closes all files opened during this session, releases all locks used during this session, and logs out the server process used to invoke the server database.

See also [“Server Sessions”](#) in Chapter 2, [“Programming with InterCall.”](#)

## Related Functions

[ic\\_opensession](#)

[ic\\_quitall](#)

[ic\\_setsession](#)

[ic\\_unidata\\_session](#)

[ic\\_universe\\_session](#)

---

# ic\_quitall

## Syntax

`ic_quitall (code)`

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_quitall Output Variable**

## Description

`ic_quitall` closes all the sessions opened by [ic\\_opensession](#), [ic\\_unidata\\_session](#), or [ic\\_universe\\_session](#). It functionally is the same as using [ic\\_quit](#) on individual sessions. All files are closed and locks are released.

See also “[Server Sessions](#)” in Chapter 2, “[Programming with InterCall](#).”

## Related Function

[ic\\_quit](#)  
[ic\\_setsession](#)

---

## ic\_raise

### Syntax

**ic\_raise** (*string*, *string\_len*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>string_len</i>	long *	Length of the string used.

**ic\_raise Input Variable**

### Input/Output Variable

The following table describes the input/output variable.

Parameter	Type	Description
<i>string</i>	char *	String in which delimiters are raised.

**ic\_raise Input/Output Variable**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_raise Output Variable**



## Description

**ic\_raise** returns *string* with its delimiters converted to the next higher-level delimiter. For example, value marks are changed to field marks, subvalue marks are changed to value marks, and so on.

## Related Function

[ic\\_lower](#)

---

# ic\_read

## Syntax

**ic\_read** (*file\_id*, *lock*, *record\_id*, *id\_len*, *record*, *max\_rec\_size*, *record\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>lock</i>	long *	Specifies the type of lock required.
<i>record_id</i>	char *	Character string containing the record ID of the record to be read.
<i>id_len</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.
<i>max_rec_size</i>	long *	Maximum record size in bytes that can be returned.

---

### ic\_read Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>record</i>	char *	Buffer that will contain the record returned by the <b>ic_read</b> function.
<i>record_len</i>	long *	Length of the record in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_read</b> is executed:

---

### ic\_read Output Variables

Parameter	Type	Description
		-1                      The user holding the lock is on a remote node.  0                        The record is not locked.  other                   The record is locked by another user. The value is the user number of the user holding the lock.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_read Output Variables (Continued)**

### Description

**ic\_read** reads a record from an open server database file.

If you do not specify a buffer large enough for the record, the buffer is filled to the value of *max\_rec\_size* and the error IE\_BTS (buffer too small) is returned.

If the record does not exist, the value of *record\_len* is set to 0 and *record* is set to null.

The following table lists the valid values for *lock*.

Lock	Meaning
IK_READ	The record is to be read without locking.
IK_READL	The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.
IK_READU	<p>The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.</p> <p>If a value of IK_READUW or IK_READLW is specified for <i>lock</i> and another user holds an exclusive lock on the record to be read, the program will pause until the lock is released.</p>
IK_READLW	The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.

***lock* Values**

Lock	Meaning
IK_READUW	<p>The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.</p> <p>If the record is already locked by another user and the value of <i>lock</i> is IK_READU, the error IE_LCK (record is locked by another user) is returned in <i>code</i>, and the user number of the user who has locked the record is returned in <i>status_func</i>. If a value of IK_READU or IK_READL is specified for <i>lock</i>, the record is locked even if <b>ic_read</b> does not execute successfully.</p> <p>The lock is released if an attempt is made to write to the record with the <a href="#">ic_write</a> or <a href="#">ic_writev</a> function. The lock also can be released by calling the <a href="#">ic_release</a> function.</p>

#### *lock* Values (Continued)

## Related Functions

[ic\\_close](#)  
[ic\\_readv](#)  
[ic\\_release](#)  
[ic\\_write](#)  
[ic\\_writev](#)

---

# ic\_readblk

## Syntax

**ic\_readblk** (*file\_id*, *text\_buffer*, *text\_len*, *block\_size*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.
<i>block_size</i>	long *	Number of characters that <b>ic_readblk</b> will read.

---

### ic\_readblk Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer that will contain the record returned by the <b>ic_readblk</b> function.
<i>text_len</i>	long *	Length of the record in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC STATUS function after <b>ic_readblk</b> is executed:
	-1	The file is not open for a read.
	0	The read is successful.
	1	The end of file is encountered, or the number of bytes passed in was <=0.

---

### ic\_readblk Output Variables



Parameter	Type	Description
<i>code</i>	long *	2 TIMEOUT ended the read.
		3 Some characters were read before a TIMEOUT ended the read.
		Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_readblk Output Variables (Continued)**

**Description**

***Note:** UniData databases do not support the **ic\_readblk** function.*

**ic\_readblk** reads a block of a set size from a server file opened for sequential processing.

**Related Functions**

- [ic\\_closeseq](#)
- [ic\\_readseq](#)
- [ic\\_seek](#)
- [ic\\_writeblk](#)
- [ic\\_writeseq](#)

---

# ic\_readlist

## Syntax

**ic\_readlist** (*select\_list\_num*, *list\_buffer*, *list\_buf\_size*, *list\_len*, *id\_count*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>select_list_num</i>	long *	Identifies the select list (0 through 10 on UniVerse systems, 0 through 9 on UniData systems) that is to be read.
<i>list_buffer</i>	char *	Buffer where the list of record IDs is to be placed.
<i>list_buf_size</i>	long *	Maximum size of the record ID list in bytes.

---

### ic\_readlist Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>list_len</i>	long *	Actual length of the record ID list in bytes.
<i>id_count</i>	long *	Number of record IDs in the returned list.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

### ic\_readlist Output Variables

## Description

**ic\_readlist** reads the remains of an active select list into a dynamic array, and copies the list of record IDs, separated by field marks, into a buffer supplied by the user. You can get an active select list by:

- Using [ic\\_getlist](#) or [ic\\_select](#)
- Using [ic\\_execute](#) to execute a UniVerse BASIC or UniBasic statement such as SELECT, which creates a select list

If you do not specify a buffer large enough for the select list, the buffer is filled to the value of *list\_buf\_size* and the error IE\_BTS (buffer too small) is returned in *code*. If the select list for the number specified is inactive, the error IE\_LRR (last record read) is returned in *code*.

## Related Functions

[ic\\_clearselect](#)

[ic\\_formlist](#)

[ic\\_readnext](#)



# ic\_readnext

## Syntax

**ic\_readnext** (*select\_list\_num*, *record\_id*, *max\_id\_size*, *id\_len*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>select_list_num</i>	long *	Select list (0 through 10 on UniVerse systems, 0 through 9 on UniData systems) from which the record ID is to be taken.
<i>max_id_size</i>	long *	Maximum size of <i>record_id</i> in bytes.

**ic\_readnext Input Variables**

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>record_id</i>	char *	Next record ID from the select list.
<i>id_len</i>	long *	Actual length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_readnext Output Variables**

## Description

**ic\_readnext** returns *record\_id* from a currently active select list. The select list is read sequentially and values of *record\_id* are returned sequentially. You can get an active select list by:

- Using [ic\\_getlist](#) or [ic\\_select](#)
- Using [ic\\_execute](#) to execute a UniVerse BASIC or UniBasic statement such as SELECT, which creates a select list

If the maximum length specified for *record\_id* is too small to receive the *record\_id* being read, the error IE\_BTS (buffer too small) is returned in *code*. If the select list is empty, the error IE\_LRR (last record read) is returned in *code*.

If the active select list is the result of an exploded select, then **ic\_readnext** writes a multivalued string containing the record ID, relative value position, and the relative subvalue position in *record\_id*. You must extract the record ID before attempting any read or write operation.

## Related Functions

[ic\\_clearselect](#)

[ic\\_formlist](#)

---

# ic\_readseq

## Syntax

**ic\_readseq** (*file\_id*, *text\_buffer*, *max\_buff*, *text\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.
<i>max_buff</i>	long *	Maximum size of <i>text_buffer</i> in bytes.

---

### ic\_readseq Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer that will contain the record returned by the <b>ic_readseq</b> function.
<i>text_len</i>	long *	Length of the record in bytes.
<i>status_func</i>	long *	Value of the UniVerse BASIC STATUS function after <b>ic_readseq</b> is executed:
	-1	The file is not open for a read.
	0	The read is successful.

---

### ic\_readseq Output Variables

Parameter	Type	Description
	1	The end of file is encountered.
	2	TIMEOUT ended the read.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_readseq Output Variables (Continued)

## Description

**Note:** UniData databases do not support the **ic\_readseq** function.

**ic\_readseq** reads a line from an open server database file which has been opened for sequential processing. Each **ic\_readseq** function reads data from the current position through the end of the line. The current position is updated to the position after the end of the line.

**Note:** UNIX uses NEWLINE (ASCII 10) as the end-of-line string. UniVerse for Windows Platforms uses the CARRIAGE RETURN-NEWLINE pair (ASCII 13-ASCII 10).

## Related Functions

[ic\\_closeseq](#)  
[ic\\_readblk](#)  
[ic\\_seek](#)  
[ic\\_writeblk](#)  
[ic\\_writeseq](#)

# ic\_readv

## Syntax

**ic\_readv** (*file\_id, lock, record\_id, id\_len, field\_number, field, max\_field\_size, field\_len, status\_func, code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>lock</i>	long *	Specifies the type of lock required: <div><div><div>IK_READ</div><div>The record is to be read without locking.</div></div><div><div>IK_READL</div><div>The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.</div></div><div><div>IK_READU</div><div>The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.</div></div><div><div>IK_READLW</div><div>The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.</div></div><div><div>IK_READUW</div><div>The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.</div></div></div>
<i>record_id</i>	char *	Character string containing the record ID of the record that contains the field to be read.

ic\_readv Input Variables

Parameter	Type	Description
<i>id_len</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.
<i>field_number</i>	long *	Number of the field from which the data is to be read.
<i>max_field_size</i>	long *	Size of the field in bytes.
<i>field_len</i>	long *	Length of the field in bytes.

#### ic\_readv Input Variables (Continued)

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>field</i>	char *	Character buffer that will contain the field that is being read.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_readv</b> is executed. If the record is not locked, 0 is returned.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_readv Output Variables

## Description

**ic\_readv** reads a specified field from a record in an open server database file.

If a value of 0 is specified for *field\_number*, **ic\_readv** checks if the record exists. The value of *field\_len* is set to 0 and *field* is set to null. If the record exists, a value of 0 is returned in *code*. If the record does not exist, the error IE\_RNF (record not found) is returned in *code*.

If you do not specify a buffer large enough for the field, the buffer is filled to the value of *max\_field\_size*, and IE\_BTS (buffer too small) is returned in *code*.

**IK\_READU:** If a value of IK\_READUW or IK\_READLW is specified for *lock* and another user holds an exclusive lock on the record to be read, the program will pause until the lock is released.

**IK\_READUW:** If the record is already locked by another user and the value of *lock* is IK\_READU, the error IE\_LCK (record is locked by another user) is returned in *code*, and the user number of the user who has locked the record is returned in *status\_func*. If a value of IK\_READU or IK\_READL is specified for *lock*, the record is locked even if **ic\_readv** does not execute successfully. The lock is released if an attempt is made to write to the record with the [ic\\_write](#) or [ic\\_writev](#) function. The lock can also be released by calling the [ic\\_release](#) function.

## Related Functions

[ic\\_close](#)

[ic\\_read](#)

[ic\\_release](#)

[ic\\_write](#)

[ic\\_writev](#)

---

## ic\_recordlock

### Syntax

**ic\_recordlock** (*file\_id*, *lock*, *record\_id*, *id\_len*, *status\_func*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>lock</i>	long *	Specifies the type of lock required:  IK_READL      The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.  IK_READU      The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.  IK_READLW     The record is locked with an IK_READL lock, preventing other users from setting an IK_READU lock. Users are able to read the record using an IK_READL lock.  IK_READUW     The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record.
<i>record_id</i>	char *	Character string containing the record ID of the record for which the lock is being taken.
<i>id_len</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.

---

#### ic\_recordlock Input Variables



# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_recordlock</b> is executed:  -1        The user holding the lock is on a remote node.  0        The record is not locked.  other    User number of the user holding the lock.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_recordlock Output Variables**

## Description

**ic\_recordlock** sets an IK\_READU lock on a record in an open server database file without performing a read.

**IK\_READU:** If a value of IK\_READUW or IK\_READLW is specified for *lock* and another user holds an exclusive lock on the record to be locked, the program will pause until the lock is released.

**IK\_READUW:** If the record is already locked by another user and the value of *lock* is IK\_READU, the error IE\_LCK (record is locked by another user) is returned in *code*, and the user number of the user who has locked the record is returned in *status\_func*. The record can be released explicitly with [ic\\_release](#) or implicitly with the [ic\\_write](#), [ic\\_writev](#), or [ic\\_delete](#) function.

## Related Functions

[ic\\_close](#)  
[ic\\_delete](#)  
[ic\\_read](#)  
[ic\\_readv](#)  
[ic\\_release](#)  
[ic\\_write](#)  
[ic\\_writev](#)

# ic\_recordlocked

## Syntax

**ic\_recordlocked** (*file\_id*, *record\_id*, *id\_length*, *lock\_status*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>record_id</i>	char *	Character string containing the record ID of the record for which the lock is being taken.
<i>id_length</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.

ic\_recordlocked Input Variables

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>lock_status</i>	long *	Lock status of the specified record.
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after the procedure has been executed:  positive value      The user number of the owner of the lock (or the first user number encountered, if more than one user has locked records in the specified file).

ic\_recordlocked Output Variables

Parameter	Type	Description
	0	The READU lock table is full.
	negative value	–1 times the user number of the remote user who has locked the record or file.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### **ic\_recordlocked Output Variables (Continued)**

### **Description**

**ic\_recordlocked** returns the status of a record lock in the *lock\_status* argument.

The following table lists the valid values for *lock\_status*.

Symbolic Constant	Value	Meaning
	4	This user has a shared FILELOCK.
LOCK_MY_FILELOCK	3	This user has an exclusive FILELOCK.
LOCK_MY_READU	2	This user has a READU lock.
LOCK_MY_READL	1	This user has a READL lock.
LOCK_NO_LOCK	0	Record not locked.
LOCK_OTHER_READL	–1	Another user has a READL lock.
LOCK_OTHER_READU	–2	Another user has a READU lock.
LOCK_OTHER_FILELOCK	–3	Another user has an exclusive FILELOCK.
	–4	Another user has a shared FILELOCK.

#### ***lock\_status* Values**

## Related Functions

[ic\\_clearfile](#)

[ic\\_delete](#)

[ic\\_release](#)

[ic\\_trans](#)

---

## ic\_release

### Syntax

**ic\_release** (*file\_id*, *record\_id*, *id\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function, or 0 if all locks are to be released.
<i>record_id</i>	char *	Character string containing the record ID of the record for which locks are to be released.
<i>id_len</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.

---

#### ic\_release Input Variables

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_release Output Variable

### Description

**ic\_release** releases records locked by the [ic\\_filelock](#), [ic\\_read](#), or [ic\\_readv](#) function, or by the UniVerse BASIC or UniBasic FILELOCK, MATREADU, READL, READU, or READVU statements.

If the value of *file\_id* is 0, **ic\_release** releases all records locked by all **ic\_filelock**, **ic\_read**, **ic\_readv**, and **ic\_recordlock** calls made during the current session.

If the value of *id\_len* is 0, all records in the specified file are released. If values are supplied for *file\_id*, *record\_id*, and *id\_len*, only locks on the specified record are released.

Use **ic\_quit** or **ic\_quitall** to release all file and record locks. If the file has been opened more than once, the locks are released on the last close.

Record locks are released automatically if **ic\_write**, **ic\_writev**, or **ic\_delete** is called.

## Related Functions

[ic\\_close](#)

[ic\\_fileunlock](#)

---

## ic\_remove

### Syntax

**ic\_remove** (*dynamic\_array*, *length\_da*, *string*, *max\_str\_size*, *string\_len*, *delimiter*, *remove\_pointer*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>length_da</i>	long *	Length of the dynamic array in bytes.
<i>max_str_size</i>	long *	Maximum size of the string buffer in bytes.

---

#### ic\_remove Input Variables

### Input/Output Variables

The following table describes the input/output variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	String containing a server database dynamic array.
<i>remove_pointer</i>	long *	Stores the position of the string which is processing.

---

#### ic\_remove Input/Output Variables



# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>string</i>	char *	Substring extracted from the dynamic array.
<i>string_len</i>	long *	Length of the new substring in bytes.
<i>delimiter</i>	long *	Value of the delimiter following the string: 0 = End of <i>string</i> 1 = Item 2 = Field 3 = Value 4 = Subvalue 5 = Text
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_remove Output Variables

# Description

**ic\_remove** assigns a substring of a dynamic array to *string*. The substring can be delimited by an item, field, value, subvalue, or text mark.

A pointer, *remove\_pointer*, is maintained to the trailing delimiter of the substring just assigned. Its initial value must be 0. Each subsequent execution moves the pointer to the next delimiter and assigns this substring to the variable. The function can be executed repeatedly until all substrings have been removed.

You can reset the pointer to 0 at any time to begin again at the start of the string, but its value must not be altered otherwise. The **ic\_remove** function does not change the value of the dynamic array.

If the string extracted from the dynamic array exceeds *max\_str\_size*, IE\_BTS (buffer too small) is returned in *code*.

## Related Functions

[ic\\_extract](#)  
[ic\\_insert](#)  
[ic\\_readlist](#)  
[ic\\_replace](#)  
[ic\\_strdel](#)

# ic\_replace

## Syntax

**ic\_replace** (*dynamic\_array*, *max\_da\_size*, *length\_da*, *field*, *value*, *subvalue*, *string*, *string\_len*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>max_da_size</i>	long *	Maximum size of the dynamic array buffer in bytes.
<i>field</i>	long *	Number of the field to be replaced.
<i>value</i>	long *	Number of the value to be replaced.
<i>subvalue</i>	long *	Number of the subvalue to be replaced.
<i>string</i>	char *	String to replace the specified substring.
<i>string_len</i>	long *	Length of the new substring in bytes.

ic\_replace Input Variables

## Input/Output Variables

The following table describes the input/output variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	String containing a server database dynamic array.
<i>length_da</i>	long *	Length of the dynamic array in bytes.

ic\_replace Input/Output Variables

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

### ic\_replace Output Variable

## Description

**ic\_replace** replaces the data content of an element of a dynamic array, returning the new dynamic array and its new length. The data content to be inserted is specified by *string*. The numeric values of *field*, *value*, and *subvalue* determine whether the new data replaces a field, value, or subvalue.

### Numeric Values of *field*, *value*, and *subvalue*

The numeric values of *field*, *value*, and *subvalue* determine whether the new data replaces a field, value, or subvalue.

- If *value* and *subvalue* are 0, the new data replaces the specified field.
- If only *subvalue* is 0, the new data replaces the specified value.
- If no argument is 0, the new data replaces the specified subvalue.
- If a higher-level argument has a value of 0, and a lower-level argument is nonzero, the 0 value becomes 1. (Field is the highest level, subvalue is the lowest.)
- If *field*, *value*, or *subvalue* evaluate to  $-1$ , the data is placed after the last field, value, or subvalue, respectively.
- If the number of characters to be added extends the length of the dynamic array past *max\_da\_size*, the original dynamic array is not altered, and the error IE\_BTS (buffer too small) is returned in *code*.

## Related Functions

[ic\\_extract](#)  
[ic\\_insert](#)  
[ic\\_readlist](#)  
[ic\\_remove](#)  
[ic\\_strdel](#)

---

## ic\_seek

### Syntax

**ic\_seek** (*file\_id*, *offset*, *relto*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	Specifies the file identifier of a file opened for sequential access.
<i>offset</i>	long *	Number of bytes before or after the reference position.
<i>relto</i>	long *	Value that specifies the reference position relative to: 0 = The beginning of the file 1 = The current position 2 = The end of the file

---

#### ic\_seek Input Variables

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_seek Output Variable

### Description

**Note:** UniData databases do not support the **ic\_seek** function.





**ic\_seek** moves a file pointer from the beginning or end of a file, relative to the current position. *offset* is the number of bytes before or after the *relto* reference point. A negative offset results in the pointer being moved before the position specified by *relto*.

**Note:** *UNIX uses NEWLINE (ASCII 10) as the end-of-line string. UniVerse for Windows Platforms uses the CARRIAGE RETURN-NEWLINE pair (ASCII 13–ASCII 10). Programs using these routines must be aware of the system type with which they are communicating, as it affects the seek position in the files.*

## Related Functions

[ic\\_closeseq](#)

[ic\\_openseq](#)

[ic\\_readblk](#)

[ic\\_readseq](#)

[ic\\_writeblk](#)

[ic\\_writeseq](#)

---

## ic\_select

### Syntax

**ic\_select** (*file\_id*, *select\_list\_num*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>select_list_num</i>	long *	Select list number (0 through 10 on UniVerse systems, 0 through 9 on UniData systems) to be used for the newly created list.

---

**ic\_select Input Variables**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

**ic\_select Output Variable**

### Description

**ic\_select** creates a list of all record IDs from an open server database file. It is similar to the UniVerse BASIC or UniBasic SELECT statement.



## Related Functions

[ic\\_clearselect](#)  
[ic\\_formlist](#)  
[ic\\_getlist](#)  
[ic\\_readlist](#)  
[ic\\_readnext](#)

---

## ic\_selectindex

### Syntax

**ic\_selectindex** (*file\_id*, *select\_list\_num*, *ak\_name*, *ak\_name\_len*, *ak\_value*, *ak\_value\_len*, *status\_func*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>select_list_num</i>	long *	Select list number (0 through 10) to be used for the newly created list.
<i>ak_name</i>	char *	Name of a secondary index on the file.
<i>ak_name_len</i>	long *	Length of <i>ak_name</i> . If <i>ak_name_len</i> is 0 or negative, a dynamic array is returned containing the secondary index names for all indexes on the file.
<i>ak_value</i>	char *	Name of a particular secondary index value for which a select list is requested.
<i>ak_value_len</i>	long *	Length of <i>ak_value</i> . An empty value is indicated by setting <i>ak_value_len</i> to 0. To get a list of all keys present in the index, <i>ak_value_len</i> should be set to a negative number.

---

#### ic\_selectindex Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	The status value. It uses the same values as the UniVerse BASIC or UniBasic STATUS function following a SELECTINDEX statement: 0 if the operation was successful or 1 if the index name supplied is not the name of a secondary index on the file.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_selectindex Output Variables

# Description

**ic\_selectindex** creates a select list based on a secondary index.

If *ak\_value\_len* is set to a negative number, **ic\_selectindex** will create a select list of all the secondary index values present in the index specified by *ak\_name*.

If *ak\_name* does not correspond to the name of a secondary index on the specified file, no select list is created, and *status\_func* is set to 1. Otherwise, *status\_func* is set to 0.

# Related Functions

- [ic\\_clearselect](#)
- [ic\\_indices](#)
- [ic\\_readlist](#)
- [ic\\_readnext](#)
- [ic\\_select](#)

---

## ic\_session\_info

### Syntax

**ic\_session\_info** (*key, data, max\_data\_len, data\_len, code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Key value specifying which piece of information is to be returned:
		IK_HOSTNAME      The name of the server currently connected.
		IK_HOSTTYPE      The type of server. A value of 1 is returned for a UNIX server and a value of 2 is returned for a Windows server.
		IK_TRANSPORT    The type of transport used to make the connection. A value of 1 is returned for a TCP/IP connection and a value of 2 is returned for a LAN pipe connection.
		IK_USERNAME      The name of the user who made the connection (unless the Microsoft Security Token was used to make the connection).
<i>max_data_len</i>	long *	IK_STATUS        This is the current state of the connection. A value greater than 0 is returned when the connection is still active, and a value of 0 is returned if the connection is down.
		Maximum length of the returned data in bytes.

---

#### ic\_session\_info Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>data</i>	char *	Buffer where the returned data is placed.
<i>data_len</i>	long *	Actual length of the returned data in bytes.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_session\_info Output Variables**

## Description

**ic\_session\_info** returns information about the current session. The information returned is determined by the *key* value.

See also “[Server Sessions](#)” in Chapter 2, “[Programming with InterCall](#).”

## Related Functions

- [ic\\_opensession](#)
- [ic\\_setsession](#)
- [ic\\_unidata\\_session](#)
- [ic\\_universe\\_session](#)

---

## ic\_set\_comms\_timeout

### Syntax

**ic\_set\_comms\_timeout** (*timeout*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>timeout</i>	long *	UniRPC timeout in seconds.

**ic\_set\_comms\_timeout Input Variable**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_set\_comms\_timeout Output Variable**

### Description

**ic\_set\_comms\_timeout** sets the UniRPC timeout in seconds for the current session. This function can be used at any time except when [ic\\_execute](#) or [ic\\_executecontinue](#) is running. In this case, the command being executed must complete before you can use **ic\_set\_comms\_timeout**.

The default timeout is 24 hours (86400 seconds). You can disable the timeout by entering a negative value or 0 for the *timeout* parameter.



**Note:** *If you enter a value for the timeout that is too small, a running command may time out (for example, if you use [ic\\_read](#)). An error code is returned, and the connection to the server is dropped.*

---

## ic\_set\_locale

### Syntax

**ic\_set\_locale** (*key*, *locale\_string*, *locale\_string\_len*, *status*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Specifies the locale information you want to set. It must be one of the following tokens: IK_LC_ALL = All categories IK_LC_TIME = Time category IK_LC_NUMERIC = Numeric category IK_LC_MONETARY = Monetary category IK_LC_CTYPE = Ctype category IK_LC_COLLATE = Collate category
<i>locale_string</i>	char *	Specifies the locale string setting for the requested category.
<i>locale_string_len</i>	long *	Specifies the length of the locale string.

---

#### ic\_set\_locale Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status</i>	long *	Contains the code IE_NLS_DEFAULT if the default locale is used.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_set\_locale Output Variables



## Description

UniData databases do not support the **ic\_set\_locale** function.

If you specify `IK_LC_ALL`, all five categories are set to *locale\_string*. If NLS mode is not enabled on the server, the error code `IE_NO_NLS` is returned in *code*. If NLS is enabled on the server, **ic\_set\_locale** does one of the following in this order:

- Looks for the specified locale string on the server in the `NLS.CLIENT.LCS` file. If it is found, uses the NLS locale name it defines.
- Looks for the default entry in the `NLS.CLIENT.LCS` file. If it is found, uses the NLS locale name it defines.
- Looks for the locale string directly to see if it is loaded. If a locale name is loaded, uses it.

See also [“UniVerse NLS in Client Programs”](#) in Chapter 1, [“Introduction.”](#)

---

## ic\_set\_map (UniVerse only)

### Syntax

**ic\_set\_map** (*map\_string*, *map\_string\_len*, *status*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>map_string</i>	char *	Specifies the map name for the current server locale.
<i>map_string_len</i>	long *	Specifies the length of <i>map_string</i> .

---

#### ic\_set\_map Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status</i>	long *	Contains the code IE_NLS_DEFAULT if the default locale is used.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_set\_map Output Variables

### Description

**Note:** *UniData databases do not support the ic\_set\_map function.*

If NLS mode is not enabled on the server, the error code IE\_NO\_NLS is returned in *code*. If NLS mode is enabled on the server, **ic\_set\_map** does one of the following in this order:



- Looks for the specified map string in the NLS.CLIENT.MAPS file on the server. If it is found, uses the NLS map name it defines.
- Looks for the default entry in the NLS.CLIENT.MAPS file on the server. If it is found, uses the NLS map name it defines.
- If the map string is DEFAULT, uses the NLSDEFSRVMAP configurable parameter.
- If an NLS map string is not found, uses the map string supplied to the routine directly.

See also [UniVerse NLS in Client Programs](#) in Chapter 1, “[Introduction.](#)”

---

## ic\_setsession

### Syntax

**ic\_setsession** (*session\_id*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>session_id</i>	long *	Session identifier obtained from an earlier call to the <a href="#">ic_opensession</a> , <a href="#">ic_unidata_session</a> , or <a href="#">ic_universe_session</a> function.

---

#### ic\_setsession Input Variable

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_setsession Output Variable

### Description

**ic\_setsession** changes the current InterCall session for the current task. The new current session opened by [ic\\_opensession](#), [ic\\_unidata\\_session](#), or [ic\\_universe\\_session](#), must be open when you call **ic\_setsession**. All InterCall functions will use the current session for that task until the current session is changed. [ic\\_quit](#) only terminates the current session; it has no effect on other sessions started by the current task. Use [ic\\_quitall](#) to close all sessions. Use [ic\\_session\\_info](#) to get information about the current session.

See also [Server Sessions](#) in Chapter 2, “[Programming with InterCall](#).”

---

## ic\_setvalue

### Syntax

**ic\_setvalue** (*key*, *data*, *data\_len*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>key</i>	long *	Key value identifying the system variable.
<i>data</i>	char *	Data to add to the system variable.
<i>data_len</i>	long *	Length of the data to add to the system variable.

---

#### ic\_setvalue Input Variables

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_setvalue Output Variable

### Description

**ic\_setvalue** sets the value of a system variable from the server program.

At this release, only one key value can be specified:

Value	Symbolic Name	System Variable
7	IK_AT_USER_RETURN_CODE	@USER.RETURN.CODE

### Related Function

[ic\\_getvalue](#)

# ic\_strdel

## Syntax

**ic\_strdel** (*dynamic\_array*, *length\_da*, *field*, *value*, *subvalue*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>field</i>	long *	Number of the field to delete.
<i>value</i>	long *	Number of the value to delete.
<i>subvalue</i>	long *	Number of the subvalue to delete.

**ic\_strdel Input Variables**

## Input/Output Variables

The following table describes the input/output variables.

Parameter	Type	Description
<i>dynamic_array</i>	char *	String that contains a server database dynamic array from which the specific field, value, or subvalue is to be deleted.
<i>length_da</i>	long *	Length of the dynamic array in bytes.

**ic\_strdel Input/Output Variables**



# Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_strdel Output Variable

# Description

**ic\_strdel** deletes an element in a dynamic array. The numeric values of *field*, *value*, and *subvalue* determine whether the data to be removed is a field, value, or subvalue.

- If *value* and *subvalue* are 0, the entire field is deleted.
- If only *subvalue* is 0, the data deleted is the specified value.
- If no argument is 0, the data deleted is the specified subvalue.

# Related Functions

[ic\\_extract](#)  
[ic\\_insert](#)  
[ic\\_remove](#)  
[ic\\_replace](#)

---

## ic\_subcall

### Syntax

**ic\_subcall** (*sub\_name*, *sub\_name\_len*, *code*, *num\_args*, *var\_args*...)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>sub_name</i>	char *	Name of the subroutine to be called.
<i>sub_name_len</i>	long *	Length of <i>sub_name</i> .
<i>num_args</i>	long *	Number of arguments used by the subroutine.

**ic\_subcall Input Variables**

### Input/Output Variable

The following table describes the input/output variable.

Parameter	Type	Description
<i>var_args</i>	ICSTRING *	List of arguments for the subroutine.

**ic\_subcall Input/Output Variable**

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_subcall Output Variable**

## Description

**ic\_subcall** calls a cataloged BASIC subroutine on the server. If the subroutine has arguments, before you use **ic\_subcall** you must:

- Define the arguments in a C variable argument list composed of ICSTRING data structures.
- Allocate initial memory for the arguments using either the [ic\\_malloc](#) or the [ic\\_calloc](#) function.

When the subroutine is executed, memory is resized dynamically to contain any returned arguments. When the subroutine is completed, release memory using the [ic\\_free](#) function.



**Warning:** *Your program will not work correctly and you may corrupt memory if you use any other function to allocate or free memory.*

Visual Basic does not support variable arguments. Visual Basic users should use the features in the UniObjects Programming Interface instead.

See also [Using the @TTY Variable](#) in Chapter 2, “[Programming with InterCall](#).”

## Example

```
char subname[4] = "TEST";
long subname_len = 4;
long code = 0;
long numargs = 3;
long size = 6;
long size2 = 7;
ICSTRING arg1, arg2, arg3;
arg1.len = 0; /* This is a null argument */
arg2.len = size;
arg2.text = ic_malloc(size);
memcpy(arg2.text, "132456", arg2.len);
arg3.len = size2;
arg3.text = ic_malloc(size2);
memcpy(arg3.text, "ACCOUNT", arg3.len);
ic_subcall(subname, &subname_len, &code, &numargs, &arg1,
&arg2, &arg3);
ic_free(arg2.text);
ic_free(arg3.text);
if(arg1.len)
{
    ic_free(arg1.text);
}
```

---

## ic\_time

### Syntax

**ic\_time** (*time*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>time</i>	long *	Server system time in internal date format.

#### ic\_time Input Variable

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_time Output Variable

### Description

**ic\_time** returns the server system time in internal format. The internal format for the time is based on a reference time of midnight, which is 0. All times are positive numbers representing the number of seconds elapsed since midnight.

### Related Functions

[ic\\_date](#)  
[ic\\_timedate](#)

---

## ic\_timedate

### Syntax

**ic\_timedate** (*string*, *max\_str\_len*, *string\_len*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>max_str_len</i>	long *	Maximum size of the returned string.

**ic\_timedate Input Variable**

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>string</i>	char *	Current server time and date.
<i>string_len</i>	long *	Length of the current server time and date.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_timedate Output Variables**

## Description

**ic\_timedate** returns the time and date in external format. The format is *hh:mm:ss dd mmm yyyy*, where:

<i>hh</i>	Hours based on a 24-hour clock
<i>mm</i>	Minutes
<i>ss</i>	Seconds
<i>dd</i>	Day
<i>mmm</i>	Month, such as jan for January
<i>yyyy</i>	Year

To return the full date and time, the *max\_str\_len* needs to be set to at least 21.

## Related Functions

[ic\\_date](#)

[ic\\_time](#)

---

## ic\_trans

### Syntax

**ic\_trans** (*filename, filename\_len, dict\_flag, record\_id, record\_id\_len, field\_no, control\_code, control\_code\_len, text\_buffer, max\_buff, text\_len, status, code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>filename</i>	char *	Name of the database file on the server.
<i>filename_len</i>	long *	Length of the file name.
<i>dict_flag</i>	long *	Indicates whether the data or dictionary records of a file are to be used.
<i>record_id</i>	char *	ID of the record to be accessed.
<i>record_id_len</i>	long *	Length of the ID to be accessed.
<i>field_no</i>	long *	Field number of the record from which the data is to be extracted.
<i>control_code</i>	char *	Specifies what action to take if data is not found or is null.
<i>control_code_len</i>	long *	Length of <i>control_code</i> .
<i>max_buff</i>	long *	Maximum size of <i>text_buffer</i> in bytes.

---

#### ic\_trans Input Variables

## Output Variable

The following table describes the output variables.

Parameter	Type	Description
<i>text_buffer</i>	char *	Buffer that contains the result of the data transfer.
<i>text_len</i>	long *	Actual length of <i>text_buffer</i> .
<i>status</i>	long *	Size of the <b>ic_trans</b> function.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

### ic\_trans Output Variable

## Description

**ic\_trans** returns the contents of a field or a record in a server database file. **ic\_trans** opens the file, reads the record, and extracts the specified data.

If *field\_no* is -1, the entire record is returned, except for the record ID. *control\_code* specifies what is done if data is not found or is the null value:

Code	Description
X	(Default) Returns an empty string if the record does not exist or data cannot be found.
C	Returns the value of <i>record_id</i> if the record does not exist or data cannot be found.
N	Returns the value of <i>record_id</i> if the null value is found.

### ic\_trans Control Codes



---

# ic\_unlock

## Syntax

**ic\_unlock** (*lock\_num*, *code*)

## Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>lock_num</i>	long *	Lock number to be cleared (0 through 63).

**ic\_unlock Input Variable**

## Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_unlock Output Variable**

## Description

**ic\_unlock** clears a public process lock. Process locks are used to protect user-defined resources or events on the server from unauthorized or simultaneous data file access by different users.

## Related Function

[ic\\_lock](#)

---

## ic\_unidata\_session

### Syntax

*session\_id* = **ic\_unidata\_session** (*server\_name*, *user\_name*, *password*, *account*, *subkey*, *status*, *unidata\_server*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>server_name</i>	char *	Name of the server to which you want to connect. See “Description” for details on how you can use this to specify the transport type to use for the connection.
<i>user_name</i>	char *	Name to use to log on to the server.
<i>password</i>	char *	Password for <i>user_name</i> .
<i>account</i>	char *	Name or path of the account to access on the server.
<i>subkey</i>	char *	Name of the device subkey, used when an application connects to a database server through a multiple-tier connection.
<i>unidata_server</i>	char *	Name of the UniData server to which you want to connect. If you do not specify <i>unidata_server</i> , the server <i>udcs</i> is opened.

---

#### ic\_unidata\_session Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>session_id</i>	long	Unique session identifier.
<i>status</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_unidata\_session Output Variables

# Description

**ic\_unidata\_session** opens a new session from the client to a UniData server and returns *session\_id*, a unique session identifier. Use the [ic\\_setsession](#) function to switch among sessions using the contents of *session\_id*.

An InterCall client can support up to 10 simultaneous sessions to different database servers.

*Note: ic\_unidata\_session does not execute the LOGIN entry.*

**ic\_unidata\_session** opens the UniData server specified by *unidata\_server*. If none is specified, it opens the server called *udcs*. These servers are defined in the *unirpcservices* file. Use **ic\_universe\_session** to open the UniVerse server *uvcs*. Use **ic\_opensession** to open the server *defcs* on either UniVerse or UniData systems.

The way in which you specify *server\_name* determines the transport type used for the connection:

- If you enter *server\_name* only, the connection is made using TCP/IP.
- If you enter *server\_name:TCP*, the connection is made using TCP/IP.
- If you enter *server\_name:LAN*, the connection is made using LAN pipes.

For TCP/IP connections, you can also specify the port number and/or the IP address to use as part of *server\_name*. For example:

- If you enter *IP address* only, a TCP/IP connection is made to the specified address.
- If you enter *server\_name:port\_number*, a TCP/IP connection is made to the specified port number on the server.



- If you enter *IP address:port\_number*, a TCP/IP connection is made to the specified port number at the given IP address.

See also the following sections in Chapter 2, “[Programming with InterCall.](#)”

- [Server Sessions](#)
- [Using the @TTY Variable](#)
- [Using the Microsoft Security Token](#)

## Related Functions

[ic\\_opensession](#)

[ic\\_quit](#)

[ic\\_quitall](#)

[ic\\_universe\\_session](#)

---

## ic\_universe\_session

### Syntax

*session\_id* = **ic\_universe\_session** (*server\_name*, *user\_name*, *password*, *account*, *subkey*, *status*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>server_name</i>	char *	Name of the server to which you want to connect. See “Description” for details on how you can use this to specify the transport type to use for the connection.
<i>user_name</i>	char *	Name to use to log on to the server.
<i>password</i>	char *	Password for <i>user_name</i> .
<i>account</i>	char *	Name or path of the account to be accessed on the server.
<i>subkey</i>	char *	Name of the device subkey, used when an application connects to a database server through a multiple-tier connection.

---

#### ic\_universe\_session Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>session_id</i>	long	Unique session identifier.
<i>status</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

---

#### ic\_universe Output Variables



## Description

**ic\_universe\_session** opens a new session from the client to a UniVerse server and returns *session\_id*, a unique session identifier. Use the [ic\\_setsession](#) function to switch among sessions using the contents of *session\_id*.

An InterCall client can support up to 10 simultaneous sessions to different database servers.

**Note:** **ic\_universe\_session** does not execute the *LOGIN* entry.

**ic\_universe\_session** always opens a server called *uvcs*, which is defined in the *unirpcservices* file. Use **ic\_unidata\_session** to open the UniData server *udcs*. Use **ic\_opensession** to open the server *defcs* on either UniVerse or UniData systems.

The way in which you specify *server\_name* determines the transport type used for the connection:

- If you enter *server\_name* only, the connection is made using TCP/IP.
- If you enter *server\_name*:TCP, the connection is made using TCP/IP.
- If you enter *server\_name*:LAN, the connection is made using LAN pipes.

For TCP/IP connections, you can also specify the port number and/or the IP address to use as part of *server\_name*. For example:

- If you enter *IP address* only, a TCP/IP connection is made to the specified address.
- If you enter *server\_name:port\_number*, a TCP/IP connection is made to the specified port number on the server.
- If you enter *IP address:port\_number*, a TCP/IP connection is made to the specified port number at the given IP address.

See also the following sections in Chapter 2, “[Programming with InterCall](#).”

- [Server Sessions](#)
- [Using the @TTY Variable](#)
- [Using the Microsoft Security Token](#)

## Related Functions

[ic\\_opensession](#)

[ic\\_quit](#)

[ic\\_quitall](#)

[ic\\_unidata\\_session](#)

---

## ic\_weofseq

### Syntax

**ic\_weofseq** (*file\_id*, *code*)

### Input Variable

The following table describes the input variable.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.

#### ic\_weofseq Input Variable

### Output Variable

The following table describes the output variable.

Parameter	Type	Description
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_weofseq Output Variable

### Description

**Note:** UniData databases do not support the **ic\_weofseq** function.

**ic\_weofseq** writes an end-of-file (EOF) mark onto a file opened for sequential processing. The EOF mark truncates the file at the current processing position.





## Related Functions

[ic\\_closeseq](#)  
[ic\\_readblk](#)  
[ic\\_readseq](#)  
[ic\\_writeblk](#)  
[ic\\_writeseq](#)

# ic\_write

## Syntax

`ic_write (file_id, lock, record_id, id_len, record, record_len, status_func, code)`

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.
<i>lock</i>	long *	Specifies what actions to perform if the record was previously locked:  IK_WRITE      Releases any locks on the target record. If another user has an exclusive lock on the target record, the write fails and a locked error is returned.  IK_WRITEW    Pauses until the lock is released if another user has an exclusive update on the target record.  IK_WRITEU    Specifies that any lock is to be retained.
<i>record_id</i>	char *	Character string containing the record ID of the record to be written.
<i>id_len</i>	long *	Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.
<i>record</i>	char *	Character string containing the record.
<i>record_len</i>	long *	Length of the record in bytes.

ic\_write Input Variables

# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_write</b> is executed:
		-3      The record failed an SQL integrity check.
		-2      The record was unlocked before the operation, and the value of lock is IK_WRITE.
		0      The record was locked before the operation, and the value of lock is IK_WRITE.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_write Output Variables

# Description

**ic\_write** writes a record to an open server database file.

# Related Functions

- [ic\\_close](#)
- [ic\\_read](#)
- [ic\\_readv](#)
- [ic\\_writev](#)

---

# ic\_writeblk

## Syntax

**ic\_writeblk** (*file\_id*, *data*, *data\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.
<i>data</i>	char *	Character string containing the data to be written.
<i>data_len</i>	long *	Length of the data in bytes.

**ic\_writeblk Input Variables**

## Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC STATUS function after <b>ic_writeblk</b> is executed.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

**ic\_writeblk Output Variables**

## Description

**Note:** UniData databases do not support the **ic\_writeblk** function.

**ic\_writeblk** writes the data held in *data* starting at the current position in a file that has been opened for sequential processing.



## Related Functions

[ic\\_closeseq](#)  
[ic\\_readblk](#)  
[ic\\_readseq](#)  
[ic\\_seek](#)  
[ic\\_writeseq](#)

---

## ic\_writeseq

### Syntax

**ic\_writeseq** (*file\_id*, *data*, *data\_len*, *status\_func*, *code*)

### Input Variables

The following table describes the input variables.

Parameter	Type	Description
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_openseq</a> function.
<i>data</i>	char *	Character string containing the data to be written.
<i>data_len</i>	long *	Length of the data in bytes.

#### ic\_writeseq Input Variables

### Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC STATUS function after <b>ic_writeseq</b> is executed:  -2     The record was unlocked before the operation.  0     The record was locked before the operation.
<i>code</i>	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

#### ic\_writeseq Output Variables



## Description

***Note:** UniData databases do not support the **ic\_writeseq** function.*

Each **ic\_writeseq** function writes *data* starting at the current position in a file that has been opened for sequential processing. An end-of-line string is written into the end of the data and the pointer is then set past the end of the line.



***Note:** UNIX uses **NEWLINE** (ASCII 10) as the end-of-line string. UniVerse for Windows NT uses the **CARRIAGE RETURN-NEWLINE** pair (ASCII 13–ASCII 10).*

## Related Functions

[ic\\_closeseq](#)

[ic\\_readblk](#)

[ic\\_readseq](#)

[ic\\_seek](#)

[ic\\_writeblk](#)

---

# ic\_writev

## Syntax

**ic\_writev** (*file\_id*, *lock*, *record\_id*, *id\_len*, *field\_number*, *field*, *field\_len*, *status\_func*, *code*)

## Input Variables

The following table describes the input variables.

Parameter	Type	Description	
<i>file_id</i>	long *	File identifier returned by the <a href="#">ic_open</a> function.	
<i>lock</i>	long *	Specifies what actions to perform if the record was previously locked:	
		IK_WRITE	Releases any locks on the target record. If another user has an exclusive lock on the target record, the write fails and a locked error is returned.
		IK_WRITEW	Pauses until the lock is released if another user has an exclusive update on the target record.
<i>record_id</i>	char *	IK_WRITEU	Specifies that any lock is to be retained.
		Character string containing the record ID of the record containing the field to be written.	
		Length of <i>record_id</i> . The maximum length of <i>record_id</i> is 255 bytes.	
<i>field_number</i>	long *	Number of the field to which the data is to be written.	
<i>field</i>	char *	Character string containing the field.	
<i>field_len</i>	long *	Length of the field in bytes.	

---

### ic\_writev Input Variables



# Output Variables

The following table describes the output variables.

Parameter	Type	Description
<i>status_func</i>	long *	Value of the UniVerse BASIC or UniBasic STATUS function after <b>ic_writev</b> is executed:  -3     The record failed an SQL integrity check.  -2     The record was unlocked before the operation, and the value of <i>lock</i> is IK_WRITE.  0      The record was locked before the operation, and the value of <i>lock</i> is IK_WRITE.  <i>code</i> long *     Either 0 if execution was successful or a specific error code if execution was not successful.

## ic\_writev Output Variables

# Description

**ic\_writev** writes a specified field to a record in an open server database file.

# Related Functions

[ic\\_close](#)  
[ic\\_read](#)  
[ic\\_readv](#)  
[ic\\_write](#)

---

# InterCall Functions by Use

# A

This appendix summarizes the InterCall functions described in detail in Chapter 3, “[InterCall Functions](#).” The functions are grouped according to use.

---

## Accessing a Server

---

Function	Description
<a href="#">ic_opensession</a>	Starts a session on a database server.
<a href="#">ic_unidata_session</a>	Starts a session on a UniData server.
<a href="#">ic_universe_session</a>	Starts a session on a UniVerse server.
<a href="#">ic_session_info</a>	Returns details about the current session.
<a href="#">ic_setsession</a>	Switch between sessions using the session identifier returned by <b>ic_opensession</b> .
<a href="#">ic_set_comms_timeout</a>	Sets the network timeout period for the current session.
<a href="#">ic_quit</a>	Closes the current session.
<a href="#">ic_quitall</a>	Closes all open sessions.

---

### Functions for Accessing a Server

---

## Reading and Modifying Records

---

Function	Description
<a href="#">ic_readv</a>	Reads a single field value from a record in an open server database file.
<a href="#">ic_read</a>	Reads a record from an open server database file.
<a href="#">ic_writev</a>	Writes a new value to a field in a record in an open server database file.
<a href="#">ic_write</a>	Writes a record to an open server database file.
<a href="#">ic_trans</a>	Returns the contents of a field or a record in an open server database file.
<a href="#">ic_release</a>	Releases a lock on one record, or all records, in a file.
<a href="#">ic_delete</a>	Deletes a record from an open server database file.

---

### Functions for Reading and Modifying Records

---

---

## Reading and Modifying Sequential Files

---

Function	Description
<a href="#">ic_openseq</a>	Opens a file for sequential processing.
<a href="#">ic_readseq</a>	Reads a line of data from a file opened for sequential processing.
<a href="#">ic_readblk</a>	Reads a block of data of a specified length from a file opened for sequential processing.
<a href="#">ic_writeseq</a>	Writes a line of data from a file opened for sequential processing.
<a href="#">ic_writeblk</a>	Writes a block of data of a specified length from a file opened for sequential processing.
<a href="#">ic_weofseq</a>	Writes an end-of-file (EOF) mark to a file opened for sequential processing.
<a href="#">ic_seek</a>	Positions a file pointer by an offset in a file opened for sequential access.
<a href="#">ic_closeseq</a>	Closes a file opened for sequential processing.

---

### Functions for Reading and Modifying Sequential Files

---

## Accessing and Modifying Strings

---

Function	Description
<a href="#">ic_oconv</a>	Converts data to an external storage format.
<a href="#">ic_iconv</a>	Converts data to an internal storage format.
<a href="#">ic_raise</a>	Raises delimiters in a dynamic array to the next higher level.
<a href="#">ic_lower</a>	Converts delimiters in a dynamic array to the next lower level.
<a href="#">ic_strdel</a>	Deletes data from a dynamic array.
<a href="#">ic_alpha</a>	Determines whether a string is alphabetic.
<a href="#">ic_extract</a>	Returns data from a dynamic array.
<a href="#">ic_fmt</a>	Formats a string in a specified pattern.
<a href="#">ic_insert</a>	Inserts data into a dynamic array.
<a href="#">ic_locate</a>	Searches a dynamic array for a string and returns a value indicating where the expression is in the array and/or where the expression should go if it is not in the array.
<a href="#">ic_remove</a>	Removes successive substrings from a dynamic array.
<a href="#">ic_replace</a>	Replaces data in a dynamic array.

---

### Functions for Accessing and Modifying Strings

---

## Accessing and Modifying Select Lists

---

Function	Description
<a href="#">ic_select</a>	Creates a select list of all the record IDs in a file.
<a href="#">ic_selectindex</a>	Creates a select list from a secondary index.
<a href="#">ic_getlist</a>	Reactivates a saved select list.
<a href="#">ic_readlist</a>	Reads an entire active select list.
<a href="#">ic_formlist</a>	Converts a dynamic array to a select list.
<a href="#">ic_readnext</a>	Returns a record ID from a currently active select list.
<a href="#">ic_clearselect</a>	Clears an active select list.

---

### Functions for Accessing and Modifying Select Lists

---

## Managing Database Files

Function	Description
<a href="#">ic_open</a>	Opens a file so that records can be read or written.
<a href="#">ic_fileinfo</a>	Returns information about a server database file.
<a href="#">ic_filelock</a>	Locks a file for exclusive use.
<a href="#">ic_fileunlock</a>	Releases the exclusive lock on a file.
<a href="#">ic_recordlock</a>	Locks a record in a file.
<a href="#">ic_recordlocked</a>	Returns the status of a locked record.
<a href="#">ic_close</a>	Closes a file opened by <b>ic_open</b> .
<a href="#">ic_clearfile</a>	Deletes all records from an open server database file.

---

### Functions for Managing Database Files





---

## Using UniVerse NLS (UniVerse Only)

***Note:** You can use the NLS functions only on UniVerse servers.*

---

Function	Description
<a href="#">ic_get_locale</a>	Retrieves the name of the locale that the server is using.
<a href="#">ic_get_map</a>	Retrieves the name of the map the server is currently using.
<a href="#">ic_get_mark_value</a>	Retrieves the character value of a UniVerse system delimiter that is used in the current character set on the server.
<a href="#">ic_set_locale</a>	Sets a locale on the server.
<a href="#">ic_set_map (UniVerse only)</a>	Sets a map name for data transfer to and from the server.

---

### Functions for Using UniVerse NLS

---

## Using System Utilities

---

Function	Description
<a href="#">ic_malloc</a>	Allocates a piece of memory.
<a href="#">ic_calloc</a>	Allocates and clears a piece of memory.
<a href="#">ic_subcall</a>	Calls a cataloged subroutine.
<a href="#">ic_unlock</a>	Clears a public process lock.
<a href="#">ic_executecontinue</a>	Resumes command execution when <b>ic_execute</b> returns IE_BTS (buffer too small).
<a href="#">ic_itype</a>	Returns the value, resulting from the evaluation of an I-descriptor.
<a href="#">ic_cleardata</a>	Flushes all data loaded in the input stack.
<a href="#">ic_free</a>	Releases previously allocated memory.
<a href="#">ic_getvalue</a>	Gets a system variable value.
<a href="#">ic_inputreply</a>	Passes data to a server at input.
<a href="#">ic_data</a>	Passes a string to a server requesting data.
<a href="#">ic_date</a>	Returns the date in internal format.
<a href="#">ic_time</a>	Returns the time in internal format.
<a href="#">ic_timedate</a>	Returns the date and time in external format.
<a href="#">ic_execute</a>	Executes a server database command.
<a href="#">ic_lock</a>	Sets a public process lock.
<a href="#">ic_setvalue</a>	Sets a system variable value.
<a href="#">ic_indices</a>	Returns information about secondary indexes on a file.

---

### Functions for Using System Utilities

---

---

# Error Codes

InterCall functions return error information as a status code. Symbolic constants representing each error number are in the files *intcall.h* (for C) and INTCALL.TXT (for Visual Basic) in the *include* subdirectory of the InterCall install directory.

If you are running an application that was developed using InterCall Release 1.0, some of the error symbols contained in your code may not be listed here, as they are no longer used in InterCall 2.5 or later. Refer to your interCALL Release 1.0 manual, or the *intcall.h* file for a full listing of these symbols and their respective codes. Only the error codes that are returned by this version of InterCall are listed here.

---

## Database Error Codes

---

Value	Symbol	Meaning
1	IE_NLS_DEFAULT	
14002	IE_ENOENT	No such file or directory
14005	IE_EIO	I/O error
14009	IE_EBADF	Bad file number
14012	IE_ENOMEM	No memory available
14013	IE_EACCES	Permission denied
14022	IE_EINVAL	Invalid argument
14023	IE_ENFILE	File table overflow
14024	IE_EMFILE	Too many open files
14028	IE_ENOSPC	No space left on device
14551	IE_BW_NETUNREACH	Network is unreachable. When you see this error, you must quit and reopen the session.
22002	IE_BTS	Buffer too small
22004	IE_LRR	Last record already read
22005	IE_NFI	File identifier given does not correspond to an open file
22009	IE_STR	FILEINFO result is a string
30001	IE_RNF	Record not found
30002	IE_LCK	File or record locked by another user
30086	IE_UFI	FILEINFO request has not been implemented
30094	IE_BIL	Bad ID length

---

Value	Symbol	Meaning
30095	IE_FIFS	File ID is incorrect for session
30096	IE_USC	Unsupported server command
30097	IE_SELFFAIL	Select failed
30098	IE_LOCKINVALID	Lock number provided is invalid
30099	IE_SEQOPENED	The file was opened for sequential access and you have attempted hashed access
30100	IE_HASHOPENED	The file was opened for hashed access and you have attempted sequential access
30101	IE_SEEKFAILED	Seek command failed
30102	IE_DATUMERROR	Internal datum error
30103	IE_INVALIDATKEY	Invalid key used for GET/SET @variables
30104	IE_INVALIDFILEINFOKEY	FILEINFO key out of range
30105	IE_UNABLETOLOADSUB	Unable to load subroutine on host
30106	IE_BADNUMARGS	Bad number of arguments for subroutine (either too many or too few)
30107	IE_SUBERROR	Subroutine failed to complete successfully
30108	IE_ITYPEFTC	I-type failed to complete correctly
30109	IE_ITYPEFAILEDTOLOAD	I-type failed to load
30110	IE_ITYPENOTCOMPILED	The I-type has not been compiled
30111	IE_BADTYPE	It is not an I-type or the I-type is corrupt
30112	IE_INVALIDFILENAME	Filename is null
30113	IE_WEOFFAILED	<b>ic_weofseq</b> failed

Value	Symbol	Meaning
30114	IE_EXECUTEISACTIVE	An execute is currently active
30115	IE_EXECUTEINOTACTIVE	An execute is currently inactive
30116	IE_BADEXECUTESTATUS	Internal execute error, execute has not returned an expected status
30117	IE_INVALIDBLOCKSIZE	Block size is invalid for call
30118	IE_BAD_CONTROL_CODE	Bad trans control code
30119	IE_BAD_EXEC_CODE	Execute did not send return codes back to client correctly
30120	IE_BAD_TTY_DUP	Failure to duplicate ttys
30124	IE_TX_ACTIVE	Transaction is active
30125	IE_CANT_ACCESS_PF	Cannot access part files
30126	IE_FAIL_TO_CANCEL	Failed to cancel an execute
30127	IE_INVALID_INFO_KEY	Bad key for <b>ic_session_info</b>
30128	IE_CREATE_FAILED	The creation of a sequential file failed
30129	IE_DUPHANDLE_FAILED	Failed to duplicate a pipe handle
31000	IE_NVR	No VOC record
31001	IE_NPN	No pathname in VOC record
33201	IE_PAR1	Bad parameter 1
33202	IE_PAR2	Bad parameter 2
33203	IE_PAR3	Bad parameter 3
33204	IE_PAR4	Bad parameter 4
33205	IE_PAR5	Bad parameter 5
33206	IE_PAR6	Bad parameter 6
33207	IE_PAR7	Bad parameter 7

Value	Symbol	Meaning
33208	IE_PAR8	Bad parameter 8
33209	IE_PAR9	Bad parameter 9
33211	IE_BSLN	Bad select list number
33212	IE_BPID	Bad partfile ID
33213	IE_BAK	Bad secondary index file
39000	IE_BAD_COMMAND	Command not recognized by server
39101	IE_NODATA	The server is not responding
39119	IE_AT_INPUT	A program executed using <b>ic_execute</b> is waiting for terminal input
39120	IE_SESSION_NOT_OPEN	The session is not opened when an action is attempted
39121	IE_UVEXPIRED	The license has expired
39122	IE_CSVERSION	Client or server is out of date; client/server functions have been updated
39123	IE_COMMSVERSION	Client or server is out of date; comms support has been updated
39124	E_BADSIG	You are trying to communicate with the wrong client or server
39125	IE_BADDIR	The directory you are connecting to does not exist or is not a database account
39126	IE_SERVERERR	An error has occurred on the server while trying to transmit an error code to the client
39127	IE_BAD_UVHOME	Unable to get the correct path to the installed database

Value	Symbol	Meaning
39128	IE_INVALIDPATH	Bad path found in UV.ACCOUNT file
39129	IE_INVALIDACCOUNT	Account name given is not an account
39130	IE_BAD_UVACCOUNT_FILE	UV.ACCOUNT file could not be found and/or opened
39131	IE_FTA_NEW_ACCOUNT	Failed to attach to the account specified
39133	IE_FTS_TERMINAL	Failed to set up the terminal for server
39134	IE_ULR	User limit has been reached
39135	IE_NO_NLS	NLS is not enabled on the server
39200	IE_SR_CREATE_PIPE_FAIL	Server failed to create the slave pipes
39201	IE_SR SOCK_CON_FAIL	Server failed to connect to socket
39202	IE_SR_GA_FAIL	Slave failed to give server the Go Ahead message
39203	IE_SR_MEMALLOC_FAIL	Failed to allocate memory for the message from the slave
39204	IE_SR_SLAVE_EXEC_FAIL	The slave failed to start correctly
39205	IE_SR_PASS_TO_SLAVE_FAIL	Failed to pass the message to the slave correctly
39206	IE_SR_EXEC_ALLOC_FAIL	Server failed to allocate the memory for the execute buffer correctly
39207	IE_SR_SLAVE_READ_FAIL	Failed to read from the slave correctly
39208	IE_SR_REPLY_WRITE_FAIL	Failed to write the reply to the slave ( <b>ie_inputreply</b> )



Value	Symbol	Meaning
39209	IE_SR_SIZE_READ_FAIL	Failed to read the size of the message from the slave
39210	IE_SR_SELECT_FAIL	Server failed to select on input channel. When you see this error, you must quit and reopen the session.
39211	IE_SR_SELECT_TIMEOUT	The select has timed out
80011	IE_BAD_LOGINNAME	Login failed (user name or password invalid)
80019	IE_BAD_PASSWORD	Password has expired
80036	IE_REM_AUTH_FAILED	Unable to set remote authorization
80144	IE_ACCOUNT_EXPIRED	The account has expired
80147	IE_RUN_REMOTE_FAILED	Unable to run as the given user
80148	IE_UPDATE_USER_FAILED	Unable to update user details
81001	UVRPC_BAD_CONNECTION	Connection is bad. When you see this error, you must quit and reopen the session.
81002	UVRPC_NO_CONNECTION	Connection is down. When you see this error, you must quit and reopen the session.
81003	UVRPC_NOT_INITED	The UniRPC has not be initialized
81004	UVRPC_INVALID_ARG_TYPE	Argument for message is not a valid type. When you see this error, you must quit and reopen the session.
81005	UVRPC_WRONG_VERSION	UniRPC version mismatch
81006	UVRPC_WRONG_VERSION	Packet message out of step. When you see this error, you must quit and reopen the session.
81007	UVRPC_NO_MORE_CONNECTIONS	No more connections available

Value	Symbol	Meaning
81008	UVRPC_BAD_PARAMETER	Bad parameter passed to the UniRPC. When you see this error, you must quit and reopen the session.
81009	UVRPC_FAILED	UniRPC failed. When you see this error, you must quit and reopen the session.
81010	UVRPC_ARG_COUNT	Bad number of arguments for message
81011	UVRPC_UNKNOWN_HOST	Bad host name, or host not responding
81012	UVRPC_FORK_FAILED	UniRPC failed to fork service correctly
81013	UVRPC_CANT_OPEN_SERV_FILE	Cannot find or open the <i>unirpcservices</i> file
81014	UVRPC_CANT_FIND_SERVICE	Unable to find the service in the <i>unirpcservices</i> file
81015	UVRPC_TIMEOUT	Connection has timed out. When you see this error, you must quit and reopen the session (start the UniRPC daemon or service on the server).
81016	UVRPC_REFUSED	Connection refused, <i>unirpcd</i> not running. When you see this error, you must quit and reopen the session.
81017	UVRPC_SOCKET_INIT_FAILED	Failed to initialize network interface
81018	UVRPC_SERVICE_PAUSED	The UniRPC service has been paused
81019	UVRPC_BAD_TRANSPORT	An invalid transport type was used

Value	Symbol	Meaning
81020	UVRPC_BAD_PIPE	Invalid pipe handle
81021	UVRPC_PIPE_WRITE_ERROR	Error writing to pipe
81022	UVRPC_PIPE_READ_ERROR	Error reading from pipe

# Glossary

client	<p>A computer system or program that uses the resources and services of another system or program (called a server).</p> <p>A process that uses resources provided by a local or remote server process. See also <b>server</b>.</p>
dynamic link library (DLL)	<p>A special executable library that applications can use to share code and resources.</p>
extended relational database	<p>A database that uses a three-dimensional file structure that supports multivalued data within nested tables, and extensible, variable-length data formats. This enables a single file (table) to contain the information that otherwise would be scattered among several interrelated files.</p>
graphical user interface (GUI)	<p>A user interface that lets users interact with a computer application using images and text.</p>
InterCall	<p>A library of functions that let a client using UNIX or Windows access data on a database server.</p>
locale	<p>(UniVerse NLS only) The language, character set, and data formatting conventions used by a group of people. In UniVerse, a locale comprises a set of conventions in specific categories (Time, Numeric, Monetary, Ctype, and Collate).</p>
select list	<p>A string of pointers to records in a file. A select list contains the record IDs of records that meet specified criteria. Select lists can be used with other database utilities such as Retrieve on UniVerse or UniQuery on UniData.</p>
server	<p>A computer running software that offers resources to clients.</p> <p>A process that accepts and handles requests from a client process.</p>
UniObjects Programming Interface	<p>An interface that allows database clients to access and manipulate data from Windows applications. It provides a bridge between a database server with its extended relational database structure and a powerful programming environment such as Visual Basic.</p>

# Index

---

## A

allocating 3-70  
     memory 3-10, 3-70

---

## C

calling  
     cataloged subroutines 3-134  
     functions from C programs 2-5  
     functions from Visual Basic  
         programs 2-6  
     cataloged subroutines, *see* subroutines  
     character mapping 1-8  
     character strings, converting to external  
         format 3-72  
     checking string types 3-9  
     clearing  
         file operations 3-12  
         files 3-12  
         input stack 3-11  
         memory 3-10  
         public process locks 3-140  
         select lists 3-14  
     clients  
         definition GL-1  
         NLS in 1-8  
     closing  
         files 3-16  
         sequential files 3-18  
         sessions 2-3  
     configurable parameters 1-8  
         NLSDEF SRVLC 1-8  
         NLSDEF SRVMAP 1-8, 3-126  
         NLSLCMODE 1-8  
         NLSMODE 1-8

connecting to Windows NT server 2-4  
     connections  
         determining type 2-7  
         LAN pipes 2-4  
     converting  
         strings to external format 3-72  
         strings to internal format 3-55  
     copying software 1-7  
     creating  
         select list of record IDs 3-115  
         select list of secondary index  
             values 3-118  
         select lists 3-40, 3-118  
     current session 2-3  
         getting information 2-3  
         quitting 3-81

---

## D

data  
     deleting in dynamic arrays 3-132  
     passing 3-20  
     writing lines 3-154  
     data blocks, writing 3-151  
     database commands, executing 3-26,  
         3-28  
     dates  
         returning in external format 3-137  
         system 3-22  
     deleting  
         dynamic array data 3-132  
         files 3-12  
         records 3-24  
     delimiters  
         lowering values 3-69  
         in NLS 3-50

- raising values 3-84
- determining connection type 2-7
- developing InterCall applications
  - requirements 1-3
- DLL, definition GI-1
- dynamic arrays 3-40
  - deleting data 3-132
  - inserting values into 3-62
  - locating 3-66
  - removing strings from 3-108
  - replacing data 3-111
  - returning data from 3-30
- dynamic buffer allocation 2-5

## E

- EOF marks, writing 3-147
- error codes 3-8, B-1
- evaluating I-descriptors 3-64
- executing database commands 3-26, 3-28
- extended relational database,
  - definition GI-1
- external format
  - converting strings to 3-72
  - time and date 3-137
- extracting fields and records 3-139

## F

- fields
  - reading 3-139
  - reading from a field 3-97
- file locks 3-100, 3-156
  - setting 3-34
- file pointers, moving 3-114
- files
  - clearing 3-12
  - closing 3-16
  - deleting 3-12
  - ictest.c* 1-5
  - ictest.mak* 1-5
  - information about 3-32
  - installation 1-4
  - intcall.h* 1-4, 1-5, B-1
  - INTCALL.TXT 1-4, 2-6, B-1
  - libuwic.a* 1-5
  - locking 3-34

- opening 3-74
- opening sequential 3-77
- reading sequential 3-95
- unirpc32.dll* 1-4, 1-7
- unlocking 3-36
- UVCLNT32.DLL 1-4, 1-7
- UVIC32.DLL 1-4, 1-7
- UVIC32.LIB 1-4
- version* 1-5
- VOC 2-7
- writing records to 3-156
- writing sequential 3-154
- &SAVEDLISTS& 3-43
- finding secondary index name 3-57
- formatting strings 3-39
- freeing memory 3-42
- functions
  - calling from C programs 2-5
  - calling from Visual Basic
    - programs 2-6

## G

- getting
  - information about current session 2-3
  - locale names 3-46
  - map names 3-47
- GUI, definition GI-1

## I

- ICSTRING type 2-5
- ictest* sample program 1-10
- ictest.c* file 1-5
- ictest.mak* file 1-5
- ic\_alpha* function 3-9
- ic\_calloc* function 3-10
- ic\_cleardata* function 3-11
- ic\_clearfile* function 3-12
- ic\_clearselect* function 3-14
- ic\_close* function 3-16
- ic\_closeseq* function 3-18
- ic\_data* function 3-20
- ic\_date* function 3-22
- ic\_delete* function 3-23
- ic\_execute* function 3-25
- ic\_extract* function 3-29
- ic\_fileinfo* function 3-31
- ic\_filelock* function 3-34
- ic\_fileunlock* function 3-36
- ic\_fmt* function 3-38
- ic\_formlist* function 3-40
- ic\_free* function 3-42
- ic\_getlist* function 3-43
- ic\_getvalue* function 3-51
- ic\_get\_locale* function 3-45
- ic\_get\_map* function 3-47
- ic\_get\_mark* function 3-49
- ic\_iconv* function 3-54
- ic\_indices* function 3-56
- ic\_inputreply* function 3-59
- ic\_insert* function 3-61
- ic\_itype* function 3-63
- ic\_locate* function 3-65
- ic\_lock* function 3-67
- ic\_lower* function 3-68
- ic\_malloc* function 3-70
- ic\_oconv* function 3-71
- ic\_open* function 3-73
- ic\_openseq* function 3-76
- ic\_opensession* function 3-78
- ic\_quit* function 3-81
- ic\_quitall* function 3-82
- ic\_raise* function 3-83
- ic\_read* function 3-85
- ic\_readblk* function 3-88
- ic\_readlist* function 3-90
- ic\_readnext* function 3-92
- ic\_readseq* function 3-94
- ic\_readv* function 3-96
- ic\_recordlock* function 3-99
- ic\_recordlocked* function 3-102
- ic\_release* function 3-105
- ic\_remove* function 3-107
- ic\_replace* function 3-110
- ic\_seek* function 3-113
- ic\_select* function 3-115
- ic\_selectindex* function 3-117
- ic\_session\_info* function 3-119
- ic\_setsession* function 3-127
- ic\_setvalue* function 3-129
- ic\_set\_comms\_timeout* function 3-121
- ic\_set\_locale* function 3-123
- ic\_set\_map* function 3-125
- ic\_strdel* function 3-131
- ic\_subcall* function 3-133

ic\_time function 3-135  
 ic\_timetype function 3-136  
 ic\_trans function 3-138  
 ic\_unidata\_session function 3-141  
 ic\_universe\_session function 3-144  
 ic\_unlock function 3-140  
 ic\_wEOFseq function 3-147  
 ic\_write function 3-149  
 ic\_writeblk function 3-151  
 ic\_writeseq function 3-153  
 ic\_writew function 3-155  
 I-descriptors, evaluating 3-64  
 information about files 3-32  
 input stack, clearing 3-11  
 inserting values into dynamic arrays 3-62  
 installing InterCall  
   files 1-4  
*intcall.h* file 1-4, 1-5, B-1  
 INTCALL.TXT file 1-4, 2-6, B-1  
 InterCall  
   applications  
     requirements for developing 1-3  
     requirements for running 1-3  
   definition G1-1  
   files 1-4  
   overview 1-2  
   sample program 1-10  
 internal format, converting strings 3-55

---

## K

key values 3-120  
   for ic\_fileinfo 3-32  
   for ic\_getvalue 3-52

---

## L

LAN Manager 1-3  
 LAN pipes connection 2-4  
*libuvic.a* file 1-5  
 locales 1-9  
   definition G1-1  
   getting names of 3-46  
   setting 3-124  
 locating string values in dynamic arrays 3-66

lock values  
   for ic\_delete 3-23  
   for ic\_read 3-86  
 locking  
   files 3-34  
   record locks 3-100  
   records 3-156  
 locks  
   clearing public process 3-140  
   file 3-97, 3-100, 3-156  
   record 3-97  
   releasing record 3-105  
   setting 3-100  
   setting public process 3-67  
   shared record lock 3-36, 3-103  
   unlocking 3-140  
   update record lock 3-36, 3-103  
 lowering delimiter values 3-69

---

## M

maps  
   getting names of 3-47  
   setting 3-126  
 memory 2-5, 3-70  
   allocating 3-10  
   clearing 3-10  
   freeing 3-42  
   releasing 3-42  
 Microsoft Security Token 2-4  
 moving file pointers 3-114

---

## N

NLS (National Language Support)  
   character mapping 1-8  
   in client programs 1-8  
   and delimiters 3-50  
   locale conversions 1-9  
 NLSDEF SRVLC parameter 1-8  
 NLSDEF SRVMAP parameter 1-8, 3-126  
 NLSLCMODE parameter 1-8  
 NLSMODE parameter 1-8  
 null value 1-9

---

## O

opening  
   files 3-74  
   sequential files 3-77  
   sessions 2-3, 3-79, 3-142, 3-145  
 output buffers, size 2-6  
 overview of InterCall 1-2

---

## P

passing  
   data 3-20  
 public process locks 3-140  
   unlocking 3-140

---

## Q

quitting  
   all sessions 3-82  
   current session 3-81

---

## R

raising delimiter values 3-84  
 reading  
   block of data from sequential files 3-89  
   fields 3-139  
   fields from records 3-97  
   from select lists 3-91  
   lines from sequential files 3-95  
   record IDs from select lists 3-93  
   records 3-86, 3-139  
 READL locks, *see* shared record locks  
 READU locks, *see* update record locks  
 record IDs, list 3-115  
 record locks  
   releasing 3-105  
   status, returning 3-103  
 records  
   deleting 3-24  
   locking 3-100  
   reading 3-86  
   reading fields 3-139  
   writing fields to 3-156  
   writing to files 3-150

- releasing
  - memory 3-42
  - record locks 3-105
- removing strings from dynamic arrays 3-108
- replacing data in dynamic arrays 3-111
- responding to server input request 3-60
- restoring select lists 3-43
- returning
  - data from dynamic arrays 3-30
  - date in external format 3-137
  - record lock status 3-103
  - secondary index information 3-57
  - session information 3-120
  - system date 3-22
  - system time 3-135
  - system variable values 3-51
  - time and date in external format 3-137
- running InterCall applications, requirements 1-3

## S

- sample program 1-10
- secondary indexes 3-118
  - finding name 3-57
  - returning information about 3-57
  - types 3-57
- security token 2-4
- select lists
  - clearing 3-14
  - creating 3-40
  - creating record ID list 3-115
  - reading from 3-91
  - reading record IDs from 3-93
  - restoring 3-43
  - secondary index values 3-118
- sequential files
  - closing 3-18
  - opening 3-77
  - reading block of data from 3-89
  - reading lines from 3-95
- server input, responding to request for 3-60
- server sessions 2-3
  - see also* sessions
- servers, definition Gl-1

- sessions
  - closing 2-3
  - current 2-3
  - information, returning 3-120
  - opening 2-3, 3-79, 3-142, 3-145
  - quitting all 3-82
  - quitting current 3-81
  - switching 2-3, 3-127
- setting
  - locales 3-124
  - locks 3-34, 3-100
  - maps 3-126
  - public process locks 3-67
  - system variable values 3-129
  - UniRPC timeout 3-121
- shared record locks 3-34, 3-97, 3-100
- size of output buffers 2-6
- status codes, *see* error codes
- string values 3-66
- strings
  - character 3-72
  - checking types of 3-9
  - converting to internal format 3-55
  - formatting 3-39
  - locating values in dynamic arrays 3-66
- subcalls
  - and ICSTRING 2-5
  - memory 3-10
- subroutines, calling 3-134
- switching sessions 2-3, 3-127
- system date 3-22
- system delimiters
  - see also* delimiters
  - and the null value 1-9
- system time 3-135
- system variables
  - returning values 3-51
  - setting values 3-129

## T

- TCP/IP connections 3-79, 3-142, 3-145
- terminating, *see* quitting
- timeouts, setting 3-121
- times
  - returning in external format 3-137

- system 3-135

## U

- UniObjects Programming Interface, definition Gl-1
- unirpc32.dll* file 1-4, 1-7
- UniRPC, setting timeout 3-121
- unlocking files 3-36
- update record locks 3-34, 3-97, 3-100
- UVCLNT32.DLL file 1-4, 1-7
- UVIC32.DLL file 1-4, 1-7
- UVIC32.LIB file 1-4

## V

- values, inserting into dynamic arrays 3-62
- variables
  - system 3-51, 3-129
  - @TTY 2-7
- version* file 1-5
- Visual Basic, calling functions from 2-6
- VOC file 2-7

## W

- writing
  - data blocks for sequential processing 3-151
  - EOF marks 3-147
  - fields to records 3-156
  - lines of data 3-154
  - records to files 3-150

## Symbols

- &SAVEDLISTS& file 3-43
- @TTY variable 2-7