InterCall Developer's Guide

UNV-111-INTR-1

Notices

Edition

Publication date: October 2010
Book number: UNV-111-INTR-1
Product version: UniVerse 11.1

Copyright

Trademarks

The following trademarks appear in this publication:

Trademark	Trademark Owner
Rocket Software™	Rocket Software, Inc.
Dynamic Connect®	Rocket Software, Inc.
RedBack®	Rocket Software, Inc.
SystemBuilder™	Rocket Software, Inc.
UniData®	Rocket Software, Inc.
UniVerse™	Rocket Software, Inc.
U2™	Rocket Software, Inc.
U2.NET™	Rocket Software, Inc.
U2 Web Development Environment™	Rocket Software, Inc.
wIntegrate®	Rocket Software, Inc.
Microsoft® .NET	Microsoft Corporation
Microsoft® Office Excel®, Outlook®, Word	Microsoft Corporation
Windows®	Microsoft Corporation
Windows® 7	Microsoft Corporation
Windows Vista®	Microsoft Corporation
Java™ and all Java-based trademarks and logos	Sun Microsystems, Inc.
UNIX®	X/Open Company Limited

The above trademarks are property of the specified companies in the United States, other countries, or both. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names as designated by the companies who own or market them.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc., are furnished under license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the copyright notice. This software and any copies thereof may not be provided or otherwise made available to any other person. No title to or ownership of the software and associated documentation is hereby transferred. Any unauthorized use or reproduction of this software or documentation may be subject to civil or criminal liability. The information in the software and documentation is subject to change and should not be construed as a commitment by Rocket Software, Inc.

Restricted rights notice for license to the U.S. Government: Use, reproduction, or disclosure is subject to restrictions as stated in the “Rights in Technical Data-General” clause (alternate III), in FAR section 52.222-14. All title and ownership in this computer software remain with Rocket Software, Inc.

Note

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Please be aware: Any images or indications reflecting ownership or branding of the product(s) documented herein may or may not reflect the current legal ownership of the intellectual property rights associated with such product(s). All right and title to the product(s) documented herein belong solely to Rocket Software, Inc. and its subsidiaries, notwithstanding any notices (including screen captures) or any other indications to the contrary.

Contact information

Rocket Software
275 Grove Street Suite 3-410
Newton, MA 02466-2272
USA
Tel: (617) 614-4321 Fax: (617) 630-7100
Web Site: www.rocketsoftware.com

Table of Contents

Appendix A InterCall Functions by Use

Appendix B Error Codes

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
Bold	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.
Italic	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.
`Courier Bold`	In examples, courier bold indicates characters that the user types or keys the user presses (for example, **`<Return>`**).
[ ]	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 **File** -> **Exit**" means you should choose **File** from the menu bar, then choose **Exit** from the File menu.
`I`	Item mark. For example, the item mark (`I`) in the following string delimits elements 1 and 2, and elements 3 and 4: `1I2F3I4V5`

Documentation Conventions

Convention	Usage
F	Field mark. For example, the field mark (**F**) in the following string delimits elements FLD1 and VAL1: FLD1**F**VAL1**V**SUBV1**S**SUBV2
V	Value mark. For example, the value mark (**V**) in the following string delimits elements VAL1 and SUBV1: FLD1**F**VAL1**V**SUBV1**S**SUBV2
S	Subvalue mark. For example, the subvalue mark (**S**) in the following string delimits elements SUBV1 and SUBV2: FLD1**F**VAL1**V**SUBV1**S**SUBV2
T	Text mark. For example, the text mark (**T**) in the following string delimits elements 4 and 5: 1**F**2**S**3**V**4**T**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** -> **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 Rocket Software'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.

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.

Chapter

1

Introduction

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
version	A text file containing the version number of InterCall.
intcall.h	A header file used when compiling InterCall application programs.
libuvic.a	An archive file used when linking InterCall application programs.
ictest.c	C source code for a sample InterCall application.
ictest.mak	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.*

Chapter

2

Programming with InterCall

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 at 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

Chapter

3

InterCall Functions

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	**ic_opensession** **ic_session_info** **ic_setsession** **ic_set_comms_timeout** **ic_quit** **ic_quitall** **ic_unidata_session** **ic_universe_session**
Reading and modifying records	**ic_read** **ic_readv** **ic_writev** **ic_write** **ic_trans** **ic_release** **ic_delete**
Reading and modifying sequential files	**ic_openseq** **ic_readseq** **ic_readblk** **ic_writeseq** **ic_writeblk** **ic_weofseq** **ic_seek** **ic_closeseq**

Functions and Their Uses

Use	Function
Accessing and modifying strings	**ic_alpha** **ic_extract** **ic_fmt** **ic_iconv** **ic_insert** **ic_locate** **ic_lower** **ic_oconv** **ic_raise** **ic_remove** **ic_replace** **ic_strdel**
Accessing and modifying select lists	**ic_select** **ic_selectindex** **ic_getlist** **ic_readlist** **ic_formlist** **ic_readnext** **ic_clearselect**

Functions and Their Uses (Continued)

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

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
string	char *	String to be tested.
string_len	long *	Length of string.

ic_alpha Input Variables

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
size	long *	Number of bytes to allocate and clear.

ic_calloc Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
ptr	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
code	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
file_id	long *	File identifier returned by the ic_open function.

ic_clearfile Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
select_list_num	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
code	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
file_id	long *	File identifier returned by the ic_open function.

ic_close Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
file_id	long *	File identifier returned by the ic_openseq function.

ic_closeseq Input Variable

Output Variable

The following table describes the output variable.

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

ic_data Input Variables

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
date	long *	Server system date in internal format.
code	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	
file_id	long *	File identifier returned by the ic_open function.	
lock	long *	Specifies what actions to perform if the user has locked the record:	
		IK_DELETE	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.
		IK_DELETEW	Pauses until the lock is released if another user has an exclusive update on the record to be deleted.
		IK_DELETEU	Retains any locks held by the user on the record after deletion.
record_id	char *	Character string containing the record ID of the record to be deleted.	
id_len	long *	Actual length of *record_id*. The maximum length of *record_id* is 255 bytes.	

ic_delete Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_delete** is executed.
code	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
command	char *	Command to be executed.
command_len	long *	Length of the command in bytes.
text_buffer	long *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
text_buf_size	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
text_len	long *	Actual length of the returned output in bytes.
return_code	long *	Value of the @SYSTEM.RETURN.CODE variable following execution of the command.
return_code_2	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 *return_code* and *return_code_2*. When @SYSTEM.RETURN.CODE has only one value, *return_code_2* is set to 0, and the value of @SYSTEM.RETURN.CODE is returned in *return_code*.
code	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
text_buf_size	long *	Maximum size of the returned output in bytes.

ic_executecontinue Input Variable

Output Variables

Parameter	Type	Description
text_buffer	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
text_len	long *	Actual length of the returned output in bytes.
return_code	long *	The value of the @SYSTEM.RETURN.CODE variable following execution of the command.
return_code_2	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, *return_code_2* is 0.
code	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_inputreply.

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
dynamic_array	char *	String containing a server database dynamic array.
length_da	long *	Length of the dynamic array in bytes.
field	long *	Number of the field to extract.
value	long *	Number of the value to extract.
subvalue	long *	Number of the subvalue to extract.
max_str_size	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
string	char *	Substring extracted from the dynamic array.
string_len	long *	Actual length of the extracted string in bytes.
code	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
key	long *	Specifies what information is required. The symbolic names for *key* are included in the intcall.h and INTCALL.TXT files.
file_id	long *	File identifier returned by a previous call to the ic_open function.
buffer_size	long *	Length of *buffer* in bytes.

ic_fileinfo Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
data	long *	Information sought or its length.
buffer	char *	Buffer where returned string information will be placed.
code	long *	Either 0 or IE_STR 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 *file_id* 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
file_id	long *	File identifier returned by the ic_open function.

ic_filelock Input Variable

Output Variables

The following table describes the output variables.

Parameter	Type	Description	
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_filelock** 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.
code	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
file_id	long *	File identifier returned by the ic_open function.

ic_fileunlock Input Variable

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_unfilelock** is executed: –2 The file was not locked before execution. 0 The file was locked before executing **ic_fileunlock**. *status_func* is always 0 for database files.
code	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
format	char *	Format pattern to use. See “Description.”
format_len	long *	Length of the format pattern in bytes.
string	char *	Input string to be formatted.
string_len	long *	Length of *string* in bytes.
max_rslt_size	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
result	char *	Buffer containing the formatted string.
result_len	long *	Length of the formatted string in bytes.
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_fmt** 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
dynamic_array	char *	Dynamic array to load as a select list.
dynamic_len	long *	Length of the dynamic array.
selnum	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
code	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
ptr	void *	Pointer to memory to be released.

ic_free Input Variable

Parameter	Type	Description
ptr	void *	Pointer to memory to be released.

Description

ic_free releases a piece of memory previously allocated by ic_calloc or ic_malloc.

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
list_name	char *	Name of the select list to be restored.
length	long *	Length of *list_name* in bytes.
select_list_num	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
code	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
key	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
max_buff_size	long *	Maximum size of the return buffer.

ic_get_locale Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
locale_string	char *	Locale string for the current server locale.
locale_string_len	long *	Length of the returned string.
code	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
max_buff_size	long *	Maximum size of the return buffer.

ic_get_map Input Variable

Output Variables

The following table describes the output variables.

Parameter	Type	Description
map_name	char *	Map name of the current server map.
map_name_len	long *	Length of the returned string.
code	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
key	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
max_buff_size	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
mark_string	char *	Character value of the system delimiter.
mark_string_len	long *	Length of the returned string.
code	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
key	long *	A key value indicating which system variable is required.
buffer_size	long *	Maximum size of *text_buffer* in bytes.

ic_getvalue Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
text_buffer	char *	Buffer that contains the value returned, as a string of characters.
text_len	long *	Length of data actually placed in *text_buffer* in bytes.
code	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*.

Value	Symbolic Name	Equivalent to	Description
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
conv	char *	Conversion code to be used.
conv_len	long *	Length of the conversion code in bytes.
string	char *	String to be converted.
string_len	long *	Length of the string to be converted in bytes.
max_rslt_size	long *	Maximum size of *result* in bytes.

ic_iconv Input Variables

Output Variables

The following table describes the output variables.

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

ic_iconv Output Variables

Parameter	Type	Description	
		1	*string* was invalid. An empty string is returned, unless *string* is the null value, in which case *string* is returned.
		2	*conv* 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
file_id	long *	File identifier returned by a previous call to ic_open.
ak_name	char *	Name of the secondary index on the file.
ak_name_len	long *	Length of *ak_name*. If *ak_name_len* is 0 or negative, a dynamic array is returned containing the secondary index names for all indexes on the file.
text_buf_size	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
text_buffer	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
text_len	long *	Actual length of the returned output in bytes.
code	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic_indicies 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
reply_string	char *	Characters that are to be supplied to the running program as terminal input.
reply_len	long *	Length of the reply string in bytes.
add_newline	long *	Indicates whether the input should be terminated with a return (NEWLINE) character. If *add_newline* is False (0), no newline is added; if *add_newline* is True (any nonzero value), the running program receives the characters of *reply_string* followed by a newline.
text_buf_size	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
text_buffer	char *	Buffer where the output from the command is to be placed. New lines in the output are replaced by field marks.
text_len	long *	Actual length of the returned output in bytes.
return_code	long *	Value of @SYSTEM.RETURN.CODE following the execution of this command.
return_code_2	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, *return_code_2* is 0.
code	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
max_da_size	long *	Maximum size of the *dynamic_array* buffer in bytes.
field	long *	Number of the field at which to insert the new string.
value	long *	Number of the value at which to insert the new string.
subvalue	long *	Number of the subvalue at which to insert the new string.
string	char *	String to insert into the dynamic array.
string_len	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
dynamic_array	char *	String containing a server database dynamic array.
length_da	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
code	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
filename	char *	Name of the server database file.
filename_len	long *	Length of the file name in bytes.
record_id	char *	Record ID of the data record to be supplied as data during the evaluation.
record_id_len	long *	Length of the data record ID in bytes.
itype_id	char *	Record ID of the I-descriptor record which is to be evaluated.
itype_id_len	long *	Length of the I-descriptor record ID in bytes.
text_buf_size	long *	Size of *text_buffer* in bytes.

ic_itype Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
text_buffer	char *	Buffer that will contain the value returned, as a string of characters.
text_len	long *	Actual length of data placed in *text_buffer* in bytes.
code	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
search	char *	Content of the field, value, or subvalue being sought.
search_len	long *	Length of the search string.
dynamic_array	char *	Dynamic array to be searched.
dynamic_len	long *	Length of the dynamic array.
field	long *	Starting field position for the search.
value	long *	Starting value position for the search.
start	long *	Field or value from which to start the search.
order	char *	String indicating the order of the elements within the dynamic array.
order_len	long *	Length of *order* is as follows: AL or A = Ascending, left-justified AR = Ascending, right-justified D = Descending, left-justified DR = Descending, right-justified

ic_locate Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
index	long *	A variable to receive the position within the dynamic array of the string being sought.
found	long *	Either 1 if *search* was found or 0 if it was not.
code	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
lock_num	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
code	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
string_len	long *	Length of the string used.

ic_lower Input Variable

Input/Output Variable

The following table describes the input/output variable.

Parameter	Type	Description
string	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
code	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
size	long *	Number of bytes to allocate.

ic_malloc Input Variable

Output Variable

The following table describes the output variable.

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

ic_oconv Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
result	char *	Buffer containing the converted string.
result_len	long *	Length of the converted string in bytes.
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_oconv** 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 *max_rslt_size*.
		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
file_id	long *	File identifier that should be used for all subsequent operations on this file.
dict_flag	long *	Indicates whether the data or dictionary file is to be opened.
filename	char *	VOC name of the file to be opened.
file_len	long *	Length of *filename* in bytes.

ic_open Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_open** is executed.
code	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 **ic_open** 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
file_id	long *	File identifier that should be used for all subsequent operations on this file.
filename	char *	VOC name of the file to be opened.
file_len	long *	Length of *filename* in bytes.
record_id	char *	Record in the file to be opened.
record_id_len	long *	Length of *record_id*.

ic_openseq Input Variables

Output Variables

The following table describes the output variables.

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

ic_openseq Output Variables

Parameter	Type	Description	
		1	*filename* is not type 1 or type 19.
		2	*filename* could not be found.
code	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
server_name	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.
user_name	char *	Name to use to log on to the server.
password	char *	Password for *user_name*.
subkey	char *	Name of the device subkey, used when an application connects to a database server through a multiple-tier connection.
account	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
session_id	long	Unique session identifier.
status	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
code	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
code	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
string_len	long *	Length of the string used.

ic_raise Input Variable

Input/Output Variable

The following table describes the input/output variable.

Parameter	Type	Description
string	char *	String in which delimiters are raised.

ic_raise Input/Output Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
file_id	long *	File identifier returned by the ic_open function.
lock	long *	Specifies the type of lock required.
record_id	char *	Character string containing the record ID of the record to be read.
id_len	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.
max_rec_size	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
record	char *	Buffer that will contain the record returned by the **ic_read** function.
record_len	long *	Length of the record in bytes.
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_read** 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.
code	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	The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record. 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_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	The record is locked with an IK_READU lock, preventing other users from setting a lock on the same record. 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_read** 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 also can be released by calling the ic_release function.

***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
file_id	long *	File identifier returned by the ic_openseq function.
block_size	long *	Number of characters that **ic_readblk** will read.

ic_readblk Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description	
text_buffer	char *	Buffer that will contain the record returned by the **ic_readblk** function.	
text_len	long *	Length of the record in bytes.	
status_func	long *	Value of the UniVerse BASIC STATUS function after **ic_readblk** 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	
		2	TIMEOUT ended the read.
		3	Some characters were read before a TIMEOUT ended the read.
code	long *	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
select_list_num	long *	Identifies the select list (0 through 10 on UniVerse systems, 0 through 9 on UniData systems) that is to be read.
list_buffer	char *	Buffer where the list of record IDs is to be placed.
list_buf_size	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
list_len	long *	Actual length of the record ID list in bytes.
id_count	long *	Number of record IDs in the returned list.
code	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
select_list_num	long *	Select list (0 through 10 on UniVerse systems, 0 through 9 on UniData systems) from which the record ID is to be taken.
max_id_size	long *	Maximum size of *record_id* in bytes.

ic_readnext Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
record_id	char *	Next record ID from the select list.
id_len	long *	Actual length of *record_id*. The maximum length of *record_id* is 255 bytes.
code	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
file_id	long *	File identifier returned by the ic_openseq function.
max_buff	long *	Maximum size of *text_buffer* in bytes.

ic_readseq Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
text_buffer	char *	Buffer that will contain the record returned by the **ic_readseq** function.
text_len	long *	Length of the record in bytes.
status_func	long *	Value of the UniVerse BASIC STATUS function after **ic_readseq** 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.
code	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	
file_id	long *	File identifier returned by the ic_open function.	
lock	long *	Specifies the type of lock required:	
		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	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.
record_id	char *	Character string containing the record ID of the record that contains the field to be read.	

ic_readv Input Variables

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

ic_readv Input Variables (Continued)

Output Variables

The following table describes the output variables.

Parameter	Type	Description
field	char *	Character buffer that will contain the field that is being read.
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_readv** is executed. If the record is not locked, 0 is returned.
code	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	
file_id	long *	File identifier returned by the ic_open function.	
lock	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.
record_id	char *	Character string containing the record ID of the record for which the lock is being taken.	
id_len	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.	

ic_recordlock Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_recordlock** 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.
code	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
file_id	long *	File identifier returned by the ic_open function.
record_id	char *	Character string containing the record ID of the record for which the lock is being taken.
id_length	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.

ic_recordlocked Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description	
lock_status	long *	Lock status of the specified record.	
status_func	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.
code	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
file_id	long *	File identifier returned by the ic_open function, or 0 if all locks are to be released.
record_id	char *	Character string containing the record ID of the record for which locks are to be released.
id_len	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.

ic_release Input Variables

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
length_da	long *	Length of the dynamic array in bytes.
max_str_size	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
dynamic_array	char *	String containing a server database dynamic array.
remove_pointer	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
string	char *	Substring extracted from the dynamic array.
string_len	long *	Length of the new substring in bytes.
delimiter	long *	Value of the delimiter following the string: 0 = End of *string* 1= Item 2 = Field 3 = Value 4 = Subvalue 5 = Text
code	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
max_da_size	long *	Maximum size of the dynamic array buffer in bytes.
field	long *	Number of the field to be replaced.
value	long *	Number of the value to be replaced.
subvalue	long *	Number of the subvalue to be replaced.
string	char *	String to replace the specified substring.
string_len	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
dynamic_array	char *	String containing a server database dynamic array.
length_da	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
code	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
file_id	long *	Specifies the file identifier of a file opened for sequential access.
offset	long *	Number of bytes before or after the reference position.
relto	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
code	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
file_id	long *	File identifier returned by the ic_open function.
select_list_num	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
code	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
file_id	long *	File identifier returned by the ic_open function.
select_list_num	long *	Select list number (0 through 10) to be used for the newly created list.
ak_name	char *	Name of a secondary index on the file.
ak_name_len	long *	Length of *ak_name*. If *ak_name_len* is 0 or negative, a dynamic array is returned containing the secondary index names for all indexes on the file.
ak_value	char *	Name of a particular secondary index value for which a select list is requested.
ak_value_len	long *	Length of *ak_value*. An empty value is indicated by setting *ak_value_len* to 0. To get a list of all keys present in the index, *ak_value_len* should be set to a negative number.

ic_selectindex Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	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.
code	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	
key	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).
		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.
max_data_len	long *	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
data	char *	Buffer where the returned data is placed.
data_len	long *	Actual length of the returned data in bytes.
code	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
timeout	long *	UniRPC timeout in seconds.

ic_set_comms_timeout Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
key	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
locale_string	char *	Specifies the locale string setting for the requested category.
locale_string_len	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
status	long *	Contains the code IE_NLS_DEFAULT if the default locale is used.
code	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
map_string	char *	Specifies the map name for the current server locale.
map_string_len	long *	Specifies the length of *map_string*.

ic_set_map Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status	long *	Contains the code IE_NLS_DEFAULT if the default locale is used.
code	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
session_id	long *	Session identifier obtained from an earlier call to the ic_opensession, ic_unidata_session, or ic_universe_session function.

ic_setsession Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
key	long *	Key value identifying the system variable.
data	char *	Data to add to the system variable.
data_len	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
code	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
field	long *	Number of the field to delete.
value	long *	Number of the value to delete.
subvalue	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
dynamic_array	char *	String that contains a server database dynamic array from which the specific field, value, or subvalue is to be deleted.
length_da	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
code	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
sub_name	char *	Name of the subroutine to be called.
sub_name_len	long *	Length of *sub_name*.
num_args	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
var_args	ICSTRING *	List of arguments for the subroutine.

ic_subcall Input/Output Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
time	long *	Server system time in internal date format.

ic_time Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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
max_str_len	long *	Maximum size of the returned string.

ic_timedate Input Variable

Output Variables

The following table describes the output variables.

Parameter	Type	Description
string	char *	Current server time and date.
string_len	long *	Length of the current server time and date.
code	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:

hh	Hours based on a 24-hour clock
mm	Minutes
ss	Seconds
dd	Day
mmm	Month, such as jan for January
yyyy	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
filename	char *	Name of the database file on the server.
filename_len	long *	Length of the file name.
dict_flag	long *	Indicates whether the data or dictionary records of a file are to be used.
record_id	char *	ID of the record to be accessed.
record_id_len	long *	Length of the ID to be accessed.
field_no	long *	Field number of the record from which the data is to be extracted.
control_code	char *	Specifies what action to take if data is not found or is null.
control_code_len	long *	Length of *control_code*.
max_buff	long *	Maximum size of *text_buffer* in bytes.

ic_trans Input Variables

Output Variable

The following table describes the output variables.

Parameter	Type	Description
text_buffer	char *	Buffer that contains the result of the data transfer.
text_len	long *	Actual length of *text_buffer*.
status	long *	Size of the **ic_trans** function.
code	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 *record_id* if the record does not exist or data cannot be found.
N	Returns the value of *record_id* 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
lock_num	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
code	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
server_name	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.
user_name	char *	Name to use to log on to the server.
password	char *	Password for *user_name*.
account	char *	Name or path of the account to access on the server.
subkey	char *	Name of the device subkey, used when an application connects to a database server through a multiple-tier connection.
unidata_server	char *	Name of the UniData server to which you want to connect. If you do not specify *unidata_server*, the server *udcs* is opened.

ic_unidata_session Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
session_id	long	Unique session identifier.
status	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
server_name	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.
user_name	char *	Name to use to log on to the server.
password	char *	Password for *user_name*.
account	char *	Name or path of the account to be accessed on the server.
subkey	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
session_id	long	Unique session identifier.
status	long *	Either 0 if execution was successful or a specific error code if execution was not successful.

ic_unverse 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
file_id	long *	File identifier returned by the ic_openseq function.

ic_weofseq Input Variable

Output Variable

The following table describes the output variable.

Parameter	Type	Description
code	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	
file_id	long *	File identifier returned by the ic_open function.	
lock	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.
record_id	char *	Character string containing the record ID of the record to be written.	
id_len	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.	
record	char *	Character string containing the record.	
record_len	long *	Length of the record in bytes.	

ic_write Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_write** 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.
code	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
file_id	long *	File identifier returned by the ic_openseq function.
data	char *	Character string containing the data to be written.
data_len	long *	Length of the data in bytes.

ic_writeblk Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC STATUS function after **ic_writeblk** is executed.
code	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
file_id	long *	File identifier returned by the ic_openseq function.
data	char *	Character string containing the data to be written.
data_len	long *	Length of the data in bytes.

ic_writeseq Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC STATUS function after **ic_writeseq** is executed: –2 The record was unlocked before the operation. 0 The record was locked before the operation.
code	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	
file_id	long *	File identifier returned by the ic_open function.	
lock	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.
record_id	char *	Character string containing the record ID of the record containing the field to be written.	
id_len	long *	Length of *record_id*. The maximum length of *record_id* is 255 bytes.	
field_number	long *	Number of the field to which the data is to be written.	
field	char *	Character string containing the field.	
field_len	long *	Length of the field in bytes.	

ic_writev Input Variables

Output Variables

The following table describes the output variables.

Parameter	Type	Description
status_func	long *	Value of the UniVerse BASIC or UniBasic STATUS function after **ic_writev** 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.
code	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

Appendix

A

InterCall Functions by Use

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
ic_opensession	Starts a session on a database server.
ic_unidata_session	Starts a session on a UniData server.
ic_universe_session	Starts a session on a UniVerse server.
ic_session_info	Returns details about the current session.
ic_setsession	Switch between sessions using the session identifier returned by **ic_opensession**.
ic_set_comms_timeout	Sets the network timeout period for the current session.
ic_quit	Closes the current session.
ic_quitall	Closes all open sessions.

Functions for Accessing a Server

Reading and Modifying Records

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

Functions for Reading and Modifying Records

Reading and Modifying Sequential Files

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

Functions for Reading and Modifying Sequential Files

Accessing and Modifying Strings

Function	Description
ic_oconv	Converts data to an external storage format.
ic_iconv	Converts data to an internal storage format.
ic_raise	Raises delimiters in a dynamic array to the next higher level.
ic_lower	Converts delimiters in a dynamic array to the next lower level.
ic_strdel	Deletes data from a dynamic array.
ic_alpha	Determines whether a string is alphabetic.
ic_extract	Returns data from a dynamic array.
ic_fmt	Formats a string in a specified pattern.
ic_insert	Inserts data into a dynamic array.
ic_locate	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.
ic_remove	Removes successive substrings from a dynamic array.
ic_replace	Replaces data in a dynamic array.

Functions for Accessing and Modifying Strings

Accessing and Modifying Select Lists

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

Functions for Accessing and Modifying Select Lists

Managing Database Files

Function	Description
ic_open	Opens a file so that records can be read or written.
ic_fileinfo	Returns information about a server database file.
ic_filelock	Locks a file for exclusive use.
ic_fileunlock	Releases the exclusive lock on a file.
ic_recordlock	Locks a record in a file.
ic_recordlocked	Returns the status of a locked record.
ic_close	Closes a file opened by **ic_open**.
ic_clearfile	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
ic_get_locale	Retrieves the name of the locale that the server is using.
ic_get_map	Retrieves the name of the map the server is currently using.
ic_get_mark_value	Retrieves the character value of a UniVerse system delimiter that is used in the current character set on the server.
ic_set_locale	Sets a locale on the server.
ic_set_map (UniVerse only)	Sets a map name for data transfer to and from the server.

Functions for Using UniVerse NLS

Using System Utilities

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

Functions for Using System Utilities

Appendix

B

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

Database Error Codes

Value	Symbol	Meaning
30095	IE_FIFS	File ID is incorrect for session
30096	IE_USC	Unsupported server command
30097	IE_SELFAIL	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	**ic_weofseq** failed

Database Error Codes (Continued)

Value	Symbol	Meaning
30114	IE_EXECUTEISACTIVE	An execute is currently active
30115	IE_EXECUTENOTACTIVE	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 **ic_session_info**
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

Database Error Codes (Continued)

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 **ic_execute** 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

Database Error Codes (Continued)

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 (**ic_inputreply**)

Database Error Codes (Continued)

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

Database Error Codes (Continued)

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 *unirpcservices* file
81014	UVRPC_CANT_FIND_SERVICE	Unable to find the service in the *unirpcservices* 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, *unirpcd* 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

Database Error Codes (Continued)

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

Database Error Codes (Continued)

Glossary

client	A computer system or program that uses the resources and services of another system or program (called a server). A process that uses resources provided by a local or remote server process. See also **server**.
dynamic link library (DLL)	A special executable library that applications can use to share code and resources.
extended relational database	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.
graphical user interface (GUI)	A user interface that lets users interact with a computer application using images and text.
InterCall	A library of functions that let a client using UNIX or Windows access data on a database server.
locale	(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).
select list	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.
server	A computer running software that offers resources to clients. A process that accepts and handles requests from a client process.
UniObjects Programming Interface	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.