CHAPTER

6

Using the Install and Licensing APIs

This chapter describes how to use the functions in the InterBase Install API as part of an application install. It includes the following topics:

- A description of the Install API and its parts, including a list of the ten entry functions
- An overview of how to use the API to write an install
- Pseudocode for a recommended install
- A reference section with details of each function
- A list of error and warning numbers and their text

About the InterBase Install API

InterBase provides developers with resources that greatly facilitate the process of installing InterBase as part of an application install on the Win32 platform. It provides mechanisms for an install that is completely silent. In addition, it allows you to interact with users if desired, to gather information from them and to report progress and messages back to them.

Using the API functions contained in **ibinstall.dll**, you can integrate the installation of your own product with the deployment of an embedded copy of InterBase. The InterBase portion of the install is *silent*: it does not display billboards and need not require intervention from the end user.

Files in the Install API

The API consists of following files:

File	Description
ibinstall.dll.	A library of functions—the "install engine" • An API that contains ten functions plus the full text of all InterBase error messages and warnings • Installed when any InterBase option is installed
ibinstall.h	For C programmers: • A header file that contains function declarations and related values, and a list of error and warning messages and their numbers • Installed with the IBDEV option
ibinstall.lib	For C programmers: • A library file that contains the list of functions in **ibinstall.dll** • Installed with the IBDEV option
ibinstall.pas	For Delphi and Pascal programmers: • An Object Pascal source file that contains function declarations and related values • Installed with the IBDEV option

TABLE 6.1 Files required for writing an InterBase install using the InterBase Install API

These files are all available on the InterBase CDROM. They are also copied as part of the InterBase install when the DEV option is chosen at install time.

What the Install API does

The functions in the InterBase Install API perform many of the steps that were previously the responsibility of the developer:

- Performs preinstall checks: check for valid operating system, correct user permissions, existing copies of InterBase, disk space, source and destination directories
- Logs all actions to a file called **ib_install.log**
- Creates the destination directory if necessary (and possible)
- Checks for option dependencies
- Copies all files, performing necessary version checks to avoid copying over newer versions
- Creates needed registry entries and increases reference count of shared files
- On NT, installs the InterBase server as a service
- Installs the InterBase Guardian in the services Manager (Windows NT) or add the Guardian to the Run section of the Registry (Windows 95/98)
- Modifies the TCP/IP **Services** file if necessary
- Writes the selected options into the uninstall file

The install handle

Each install instance has a unique handle that identifies it. This handle is a variable of type *OPTION_HANDLE* (see page "Datatypes defined for the Install API" on page 36) that you initialize to zero at the beginning of the InterBase install. Throughout this chapter, this variable is referred to as *handle*, and its address is *phandle*. You are, of course, free to name it what you will. Once you have passed it to *isc_install_set_option*(), it references a data area where all the options for the current install are stored. You need not and should not dereference *handle* directly. The install data is all maintained by the install engine. You need only pass *handle* or a pointer to it—depending on the syntax of the function you are calling—and the install engine does all the work for you.

You must pass *handle* to *isc_install_set_option*() before passing it to any of the other functions, since *isc_install_set_option*() is the only function that accepts *handle* when its value is zero. The others return an error.

Error handling

Each of the functions in the InterBase Install API returns an error number as follows:

- If the function executes successfully, it returns zero (*isc_install_success*).
- If it completes with warnings, it returns a negative number that corresponds to a specific warning message.
- If an error occurs, it returns a positive number that corresponds to a specific error message.

You should check the return each time you call a function. If the return is nonzero, call *isc_install_get_message*() to get the text of the error or warning. For example:

```
error = isc_install_precheck(handle, source_path, dest_path)
    if(error)
    isc_install_get_message(error, message, length(message))
```

The steps in "Overview of the process" do not explicitly remind you to do this. It is assumed that you will do so as necessary.

Callback functions

The *isc_install_execute*() and *isc_uninstall_execute*() functions permit you to pass in pointers to an error-handling function and to a status function, both of which are supplied by you.

- The status function can pass status information to the end user and pass back a "cancel" request from the user.
- You can use the error-handing function to specify a response to an error or warning and to display message text to the end user.

The syntax of these functions must be as follows:

***fp_status*()**

```
int (*fp_status)(int status, void *status_arg, const TEXT* act_desc)
```

fp_status() is a callback function supplied by you, the developer, that accepts an integer, *status*, from 0 to 100, indicating percent of install/uninstall completed. When a pointer to this function is passed to either *isc_install_execute*() or *isc_uninstall_execute*(), it calls *fp_status*() at intervals and passes it a number indicating percent completion so that you can display a status bar or other indicator to the end user.

The *act_desc* parameter provides text that can be displayed as part of the progress indicator.

status_arg is a pointer to optional user-defined data passed to *isc_install_execute*() or *isc_uninstall_execute*().

Return Value The *fp_status*() function must return either 0 (continue) or 1 (abort).

***fp_error*()**

```
int (*fp_error)(MSG_NO msg_no, void *status_arg, const TEXT* context)
```

fp_error() is a callback function that accepts an error number, *msg_no*, when a pointer to it is passed to either *isc_install_execute*() or *isc_uninstall_execute*() as a parameter.

status_arg is a pointer to optional user-defined data passed to *isc_install_execute*() or *isc_uninstall_execute*().

The *context* parameter provides additional information about the nature of the error that can be passed on to the end user.

Return Value *fp_error*() processes the error message and returns one of three values: *isc_install_fp_retry*, *isc_install_fp_continue*, or *isc_install_fp_abort*.

fp_error() returns	Effect on calling function
abort	Function fails
retry	Function is called again
continue	Function ignores the error and continues from the point where the error occurred

IMPORTANT These callback functions can make calls only to *isc_install_get_message*(). The result is undetermined if they attempt to call any other Install API function.

Datatypes defined for the Install API

The following datatypes are defined for the Install API functions:

Datatype	Definition
OPTIONS_HANDLE	void*
TEXT	char
MSG_NO	long
OPT	unsigned long
FP_STATUS	function pointer of type int (*fp_status)(int status, void *status_arg, const TEXT* description)
FP_ERROR	function pointer of type int (*fp_error)(MSG_NO msg_no, void *status_arg, const TEXT* description)

TABLE 6.2 Datatypes defined for the InterBase Install API

Writing an InterBase install

"Overview of the process" below lists the steps to follow and issues to consider when writing an InterBase install. The steps you use depend on whether you are writing a silent install or an interactive install. Some steps are merely recommended rather than required, such as Calling *isc_clear_options*() before proceeding with the rest of the install. Others vary depending on whether you are also performing tasks such as writing an uninstall program, creating icons, adding authorization codes, and starting the server.

Following the list of functions there is pseudocode that provides much of the same information in code form.

IMPORTANT There must be only one InterBase server per machine. Although there are ways around this, they are not recommended. It is particularly important to avoid putting a SuperServer version of InterBase (V 5.0 and later) on a machine where a Classic server is still installed.

Overview of the process

1. The files that you need to develop and compile your application are in the ***ib_install_dir*\SDK** directory if you installed InterBase on your development system with the IB_DEV option. They are also on the InterBase CDROM in the **\SDK** directory. Collect the following files:

   - For C/C++ programmers: **ibinstall.dll**, **ibinstall.lib**, **ibinstall.h**

   - For Delphi programmers: **ibinstall.dll**, **ibinstall.pas**

   Place **ibinstall.dll** in the directory that will contain your executable after it is compiled. Place the other files where your compiler can find them.

2. Declare a variable of type *OPTIONS_HANDLE* for *handle* and initialize it to 0 (a long INT). If you are writing a companion uninstall program, allocate an array buffer for the uninstall file name.

3. If you need messages in a language other than English, call *isc_load_external_text*() to load the error and warning messages.

4. *For interactive installs only* The next steps temporarily select a group of options in order to check that there is a valid operating system, that no Classic server is present, and that there is no InterBase server running. This prevents the case where the end user answers several questions and then finds that the install cannot be performed because of an invalid OS or the presence of the Classic server:

   - Call *isc_install_set_option*() with the following parameters:

   ```
   isc_install_set_option(handle, INTERBASE)
   ```

   If you will be installing a client but no server, substitute *IB_CLIENT* for *INTERBASE*.

   - Call *isc_install_precheck*(*handle*, NULL, NULL)

   - Call *isc_install_clear_options*().

5. In an interactive install, query users for a destination and desired options.

6. Call *isc_install_set_option*() once for each option to install. In an interactive GUI setup, this would typically be invoked by a mouse click in a check box.

7. Call *isc_install_precheck*() a second time. This time, provide the source and destination path and selected options. *isc_install_precheck*() checks that the destination directory exists and is writable. If the directory does not exist and cannot be created, the function exits with an error. It also checks the dependencies of the selected options and issues a warning if the selections are incompatible or require options not selected. See page 45 for a further description of this function.
8. Call *isc_install_execute*(), passing in *handle*, the source path, and the destination path. If you have written functions to handle errors and display status, you pass in pointers to these functions and optionally pointers to context data as well. The last parameter is an optional pointer to a buffer where the uninstall file name can be stored. If you are providing a companion uninstall program, you must declare a text buffer for the name of the uninstall file and pass in a pointer to it as the final argument for this function. *isc_install_execute*() then performs the actual install.

The next steps are all optional.

9. When the install is complete, you can enable licensed functionality for the product by calling functions in the Licensing API (**iblicense.dll**) and providing certificate IDs and keys. If you do not do this, the end user must enter certificate ID and key pairs (authorization codes) before starting the server.
10. Create shortcuts on the Start menu.
11. Start the server. You can do this only after providing valid certificate IDs and keys.

A real-world example

The source code for the InterBase **setup.exe** is in ***ib_install_dir*\Examples\Install**. Since it uses the InterBase Install API, it serves as an example of how to make use of the entry points to write an install program.

The Install API functions

The InterBase Install API, **ibinstall.dll**, is a library of functions that facilitate the process of installing and deploying InterBase as part of the developer's own application. The table below lists each entry point in **ibinstall.dll** with a brief description.

Following the table is a list of datatypes that are defined for these functions and a detailed description of each function.

Function	Description
isc_install_clear_options()	Clears all options set by *isc_install_set_option*()
isc_install_execute()	Performs the actual install, including file copying, registry entries, saving uninstall options, and modifying the Services file if necessary
isc_install_get_info()	Returns the requested information in human-readable form: a suggested install directory, required disk space, an option name, or option description
isc_install_get_message()	Returns the text of the requested error or warning message number
isc_install_load_external_text()	Loads the messages from the specified message file
isc_install_precheck()	Performs a number of necessary check on the install environment, such as checking for existing servers, disk space and access, user permissions, and option dependencies
isc_install_set_option()	Creates a handle to a list of selected install options; must be called once for each option
isc_install_unset_option()	Removes an option from the list of selected options obtained from *isc_install_set_option*()
isc_uninstall_execute()	Removes installed InterBase files (but see exceptions on page 50), updates the registry, removes shares files that have a reference count less than 1, uninstalls the InterBase Guardian and Server services
isc_uninstall_precheck()	Checks for running server, correct user permission, and validity of the uninstall file

TABLE 6.3 Entry points in **ibinstall.dll**

isc_install_clear_options()

Syntax

```
MSG_NO isc_install_clear_options(OPTIONS_HANDLE *phandle)
```

Parameter	Type	Description
phandle	OPTIONS_HANDLE*	• Pointer to the handle of the list of options for the current install • You must initialize this to zero before first use • Handle is maintained by the install engine; you do not need to and should not dereference it

Description *isc_install_clear_options*() clears all the options and other install data stored in *handle* and sets *handle* to zero. It returns a warning if *handle* is zero.

It is good practice to call this function both at the beginning and at the end of an install to free all resources. After calling *isc_install_clear_options*(), you must pass *handle* to *isc_install_set_option*() at least once before passing it to any of the other install functions.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_install_execute()

Syntax

```
MSG_NO isc_install_execute(OPTIONS_HANDLE handle, TEXT *source_path,
   TEXT *dest_path, FP_STATUS *fp_status, void *status_arg,
   FP_ERROR *fp_error, void *error_arg, TEXT *uninst_file_name )
```

Parameter	Type	Description
handle	OPTIONS_HANDLE	• The handle to the list of options created by *isc_install_set_option*() • *isc_install_execute*() returns an error if the value of *handle* is NULL or zero
source_path	TEXT*	• The path where the files to be installed are located, typically located on a CDROM • *isc_install_execute*() returns an error if *source_path* is NULL or an empty string
dest_path	TEXT*	• The path to the desired install location • *isc_install_execute*() returns an error if *dest_path* is NULL or an empty string
fp_status	FP_STATUS*	• A pointer to a callback function that accepts an integer from 0 to 100; see page 34 for more information • May be NULL if no status information is required by the end user
status_arg	void*	• User-defined data to be passed to *fp_status*() • Value is often NULL
fp_error	FP_ERROR*	• A pointer to a callback function that accepts an error number and returns a mnemonic specifying whether *isc_install_execute*() should abort, continue, or retry
error_arg	void*	• User-defined data to be passed to *fp_error*() • Value is often NULL
uninst_file_name	TEXT*	• A pointer to a buffer containing the name of the uninstall file • Can be set to NULL

Description *isc_install_execute*() performs the actual install, including the following operations:

- Calls *isc_install_precheck*() to ensure that the install can be performed; if *isc_install_precheck*() returns an error the install aborts
- Logs all actions to a temporary file called **ib_install.log**
- Creates the destination directory if it does not already exist
- Copies the files using all the correct version checks and delayed copying methods if necessary
- Creates the required registry entries
- Increments UseCount entries in the Registry for shared files
- Installs the Guardian and Server as services on Windows NT, or adds the Guardian to the Run section of the registry on Windows 95
- If necessary, adds *gds_db* to the Services file
- Streams the selected options into **ib_uninst.*nnn*** (where *nnn* is a sequence number) for use at uninstall time
- Frees the options list from memory
- Upon completion, moves **ib_install.log** to the install directory
- Calls *fp_status*() at regular intervals to pass information on the install progress (percent complete)
- Attempts to clean up if at any point in the install is canceled by the user or by an error

If you choose to write functions for displaying status and handling errors, you pass in pointers to these functions as the *fp_status* and *fp_error* parameters. In addition, you can pass context information or data to these functions by passing in values for *status_arg* and *error_arg*, although these last two parameters are more commonly NULL. See page 34 for more about these callback functions.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_install_get_info()

Syntax

```
MSG_NO isc_install_get_info(OPT option, int info_type, void *info_buf,
    unsigned int buf_len)
```

Parameter	Type	Description	
info_type	int	Specifies the type of information requested; can be any one of the following values	
		isc_install_info_destination (value = 1)	• Returns a suggested destination • Ignores any value passed for *option*
		isc_install_info_opspace (value = 2)	• Returns the disk space required to install a particular option • Requires a valid value for *option*
		isc_install_info_opname (value = 3)	• Returns a human-readable option name for the specified option • Requires a valid value for *option*
		isc_install_info_opdescription (value = 4)	• Returns a human-readable description for the specified option • Requires a valid value for *option*
option	OPT	• The option for which information is requested if *info_type* is 2 through 4 • Returns an error if option is not one of the valid types listed under *isc_install_set_option*(), page 47	
info_buf	void*	• *isc_install_get_info*() writes the requested information to this buffer • Returns an error if *info_buf* is NULL • If disk space information is requested, the result is an unsigned long	
buf_len	unsigned int	• The length in bytes of *info_buf* • Returns an error if *buf_len* is NULL • Value should be al least ISC_INSTALL_MAX_MESSAGE_LEN bytes • If a destination suggestion is requested, the recommended buffer size is ISC_INSTALL_MAX_PATH	

Description *isc_install_get_info*() returns the information requested by *info_type* into *info_buf* location. The *info_buf* and *buf_len* parameters cannot be NULL.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

The contents of *info_buf* are undetermined if *isc_install_get_message*() returns anything other than zero, so the caller should always check the return from this function.

isc_install_get_message()

Syntax

```
MSG_NO isc_install_get message(MSG_NO msg_no, TEXT *msg, int msg_len)
```

Parameter	Type	Description
msg_no	MSG_NO	• Message number for which text is requested • This is the return from all the Install API functions
msg	TEXT*	• A pointer to the buffer in which the message will be returned • The message is always null-terminated
msg_len	int	• The length of *msg* in bytes • Value should be al least ISC_INSTALL_MAX_MESSAGE_LEN bytes

Description *isc_install_get_message*() converts the error or warning value stored in *msg_no* and returns the corresponding message text to the programmer.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_install_load_external_text()

Syntax

```
MSG_NO isc_install_load_external_text(TEXT *external_path)
```

Parameter	Type	Description
external_path	TEXT*	A pointer to a buffer that contains the full path and filename of a file containing error and warning messages in a language other than English

Description *isc_install_load_external_text*() loads the message file from the named path. This file contains the text of the install error and warning messages as well as option names and descriptions, and action text, description, and status messages.

If you are using English-language messages, there is no need to call this function. For messages in other languages, you can purchase translations from some InterBase VARs and use this function to load them. You must initialize this buffer with the path and filename to use.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

isc_install_precheck()

Syntax

```
MSG_NO isc_install_precheck(OPTIONS_HANDLE handle, TEXT *source_path,
    TEXT *dest_path)
```

Parameter	Type	Description
handle	OPTIONS_HANDLE	• The handle to the list of options created by *isc_install_set_option*() • *isc_install_precheck*() returns an error if the value of *handle* is NULL or zero
source_path	TEXT*	• The path where the files to be installed are located, typically located on a CDROM • Can be NULL
dest_path	TEXT*	• The path to the desired install location • Can be NULL

Description *isc_install_precheck*() performs the following checks to ensure that installation is possible:

- Checks for a valid operating system. These are currently Windows 95, Windows 98, and Windows NT 4.
- Checks that a Classic server is not present. The InterBase server (SuperServer) is a multithreaded architecture and cannot coexist with the Classic server.
- Checks that *source_path* exists and is a directory readable by the user. No check is performed if *source_path* is NULL or an empty string.
- Checks that *dest_path* is a directory writable by the user or that the directory can be created and that the drive contains enough space to install the selected components. No check is performed if *dest_path* is NULL or an empty string.
- If the *IB_SERVER* option is specified, checks whether any existing newer or older version of the SuperServer is already running.
- On NT, if the *IB_SERVER* option is specified, checks that the user performing the install has administrative privileges.
- Checks the dependencies of the options required. The dependencies between the options are as follows:

If any of these are specified	These must also be installed
IB_CMD_TOOLS, *IB_GUI_TOOLS*, *IB_DEV*, and *IB_ODBC*	*IB_CLIENT*
IB_EXAMPLES	*IB_SERVER*, *IB_CLIENT*, and *IB_DEV*
IB_EXAMPLE_API	*IB_CLIENT* and *IB_DEV*
IB_EXAMPLE_DB	*IB_SERVER*

isc_install_precheck() returns an error if any of the checks besides option dependencies fails. It returns a warning if necessary options have not been specified.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_install_set_option()

Syntax

```
MSG_NO isc_install_set_option(OPTIONS_HANDLE *phandle,
    OPT option)
```

Parameter	Type	Description
phandle	OPTIONS_HANDLE*	• Pointer to the handle of the list of options for the current install • You must initialize this to zero before first use • Handle is maintained by the install engine; you do not need to and should not dereference it
option	OPT	*option* can be any one of the following values:
		INTERBASE — • Installs all interbase components and their related files; same as specifying IB_SERVER, IB_CLIENT, IB_CMD_TOOLS, IB_GUI_TOOLS, IB_DOC, IB_EXAMPLES, and IB_DEV
		IB_SERVER — • Installs the Server component of InterBase: the server, the license file if present, the message file, the Guardian, the server configuration tool, **gstat**, **gds_lock_print**/**iblockpr**, the UDF library, the international character set library, and the help files • Makes all necessary additions to the registry • Creates the InterBase service on NT • On NT, modifies the Services file, if necessary, to add the gds_db service
		IB_CLIENT — • Installs the Client component of InterBase: the client library, the license file, and the message file • Makes all necessary additions to the registry (Windows only) • On NT, modifies the Services file, if necessary, to add the gds_db service
		IB_CMD_TOOLS — • Installs all the command line tools for InterBase on Windows platforms: **gbak**, **gfix**, **gsec**, **gstat**, **iblockpr**, and **isql** • Issues a warning if the IB_CLIENT option has not been specified

Parameter Type	Description	
	IB_GUI_TOOLS	• Installs all the InterBase GUI tools and their related help files • Issues a warning if the IB_CLIENT option has not been specified
	IB_DOC	• Installs the InterBase documentation in Adobe Acrobat PDF form
	IB_EXAMPLES	• Installs all the InterBase examples; has the same effect as specifying IB_EXAMPLE_API and IB_EXAMPLE_DB • Issues a warning if the IB_SERVER, IB_CLIENT, and IB_DEV options have not been specified
	IB_EXAMPLE_API	• Installs all API, SQL, DSQL, and ESQL example files • Issues a warning if the IB_CLIENT and the IB_DEV options have not been specified
	IB_EXAMPLE_DB	• Installs all example databases • Issues a warning if the IB_SERVER option has not been specified
	IB_DEV	• Installs the development tools and files for InterBase: **gpre**, the import libraries, and the header files
	IB_ODBC	• Installs the ODBC driver and the ODBC 3.0 driver manager

Description *isc_install_set_option*() creates and maintains a handle to a list of requested option values. You must call *ib_install_set_option*() once for each option to be installed. In an interactive install, the function would typically be invoked by a mouse click in a check box.

You must initialize *handle* to zero before calling *isc_install_set_option*() for the first time.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_install_unset_option()

Syntax

```
MSG_NO isc_install_unset_option(OPTIONS_HANDLE *phandle, OPT option)
```

Parameter	Type	Description
phandle	OPTIONS_HANDLE	• Pointer to the handle of the list of options for the current install • You must initialize this to zero before first use • Handle is maintained by the install engine; you do not need to and should not dereference it
option	OPT	• *option* can be any of the values listed for *isc_install_set_option*() • If *option* is the only member of the list, sets *handle* to zero

Description *isc_install_unset_option*() removes the option specified by *option* from the list maintained by *handle*. You must call this function once for each option to be removed. If *handle* is zero when this function is called, the function generates a warning.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_uninstall_execute

Syntax

```
MSG_NO isc_uninstall_execute(TEXT *uninstall_file_name,
    FP_STATUS *fpstatus, void *status_arg, FP_ERROR *fp_error,
    void *error_arg)
```

Parameter	Type	Description
uninstall_file_name	TEXT*	• The name of the file containing the options that were installed • Cannot be NULL
fp_status	FP_STATUS*	• A pointer to a callback function that accepts an integer from 0 to 100; see page 34 for more information • May be NULL if no status information is required by the end user
status_arg	void*	• User-defined data to be passed to *fp_status*() • Value is often NULL
fp_error	FP_ERROR*	• A pointer to a callback function that accepts an error number and returns a mnemonic specifying whether *isc_install_execute*() should abort, continue, or retry
error_arg	void*	• User-defined data to be passed to *fp_error*() • Value is often NULL

Description *isc_uninstall_execute*() performs the actual uninstall, including the following steps:

- Calls *isc_uninstall_precheck*() to ensure that the uninstall can be performed.
- Decrements UseCount entries in the Registry for shared files and remove any files that have a reference count less than one, except for files that have a value of zero preassigned by Microsoft, such as **msvcrt.dll**.
- Removes all InterBase files named in **ib_uninst.*nnn*** except for **isc4.gdb**, **isc4.gbk**, and **ib_license.dat**.
- Removes all registry entries in **ib_uninst.*nnn***.
- On Windows NT, uninstalls the Guardian and Server services. On Windows 95/98, removes the Run registry entries for them.
- Calls *fp_status*() at regular intervals to keep caller informed of uninstall status.
- Cleans up if uninstall is cancelled by the user if by an error.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

isc_uninstall_precheck()

Syntax

```
MSG_NO isc_uninstall_precheck(TEXT *uninstall_file_name)
```

Parameter	Type	Description
uninstall_file_name	TEXT*	• A pointer to the name of the uninstall file that was created by *isc_install_execute*() • Cannot be NULL

Description *isc_uninstall_precheck*() performs several checks to determine that an uninstall is possible. It checks that:

- The operating system is valid: Windows NT4, 98, or 95
- The uninstall file (**ib_uninst.*nnn***) is valid and contains the streamed list of options.
- The server, if installed, is not running.
- The user performing the uninstall is a member of either the administrator or poweruser groups when the platform is Windows NT; no equivalent check is performed on Windows 95/98.

Example To come after beta@@

Return Value Returns zero if the function executes successfully, a positive number if an error occurs, and a negative number if the function completes with warnings.

Call *isc_install_get_message*() to obtain the error message when the result is nonzero.

Call *isc_install_get_message*() to obtain the text of an error message or warning when the result of one of the Install API functions is nonzero.

Using the License API

The InterBase server functionality must be activated by installing *authorization codes* that are provided on software activation certificates obtained from InterBase. Each authorization code consists of a Certificate ID and Certificate key. You can activate the server as part of your install by using functions provided in the InterBase License API. If you do not activate the server as part of the install, it will be inactive until the end user provides authorization codes using IBConsole.

The InterBase License API (**iblicense.dll**) provides five functions that allow you to check, add, remove, and view certificate ID and key pairs (authorization codes). The fifth function retrieves and displays messages associated with the return values from the other four functions.

Loading the License API

You cannot statically load **iblicense.dll** during an install process. Use the Windows *LoadLibrary*() API call or other language-specific equivalent to load it dynamically when you need it and free the library immediately after use.

Typically, you would load the License API at the beginning of an install in order to check that your desired certificate ID/key pairs can indeed be added. Call *isc_license_check*() and then free the library. Later, when you have completed the install portion and are ready to add authorization codes, load **iblicense.dll** again and add the authentication codes. This sequence avoids the case in which an install is completed and then must be uninstalled because the authentication codes cannot be added for some reason.

Preparing the ib_license.dat file

The authorization codes are stored in the **ib_license.dat** file in the InterBase root directory. This file contains authorization codes from previous installs, which will still be functional with the current install. Each additional authorization code adds further functionality to the server. There is also an **ib_license.dat** file on the InterBase CD-ROM, which contains the client activation code for the current client version. After completing your install, follow these steps to ensure that you are using the most recent client authorization and that no prior authorization codes are lost:

- Check for the existence of **ib_license.dat** in the InterBase install directory.
- If the file is found, concatenate it with the **ib_license.dat** that is on the CD-ROM to add the current client capability.

- If the file is not found, copy **ib_license.dat** from the CD-ROM to the InterBase install directory.

These steps ensure that you have retained any existing licensed server functionality while providing functionality for the latest client.

The capabilities activated on the server are the union of the capabilities activated by each line

Adding server functionality

There are five functions available for adding and removing authorization codes in **ib_license.dll**:

- *isc_license_add*() adds a line to **ib_license.dat**. Use only authorization codes that you have been given expressly as deployment codes from Inprise Corp.
- *isc_license_check*() checks to see whether an authorization code *could* be added to **ib_license.dat**. This function performs all the same tasks as *isc_license_add*(), without actually modifying **ib_license.dat**.
- *isc_license_remove*() removes a line from **ib_license.dat**.
- *isc_license_display*() displays the authorization codes that are currently in **ib_license.dat**.
- *isc_license_get_msg*() returns the text of error messages that correspond to error codes returned by the other four licensing functions.

isc_license_add()

Syntax `int isc_license_add(char *cert_id, char *cert_key)`

Parameter	Type	Description
cert_id	char*	Pointer to a NULL-terminated character buffer containing the certificate ID to be added
cert_key	char*	Pointer to a NULL-terminated character buffer containing the certificate key to be added

Description Adds a line containing the specified certificate ID and key pair to the **ib_license.dat** file in the InterBase install directory. This ID/key pair must be a valid authorization code obtained from Inprise sales. InterBase typically requires several authorization codes to run and you must call the function once for ID/key pair you need to add.

Example To come after beta@@

Return Value *isc_license_add*() returns *isc_license_msg_restart* if it successfully adds the authorization code. If it returns an error, pass the return value to *isc_license_get_msg*() to obtain the exact error message. The possible return values are:

Return	Description
isc_license_msg_restart	Authorization code was successfully added
isc_license_msg_writefailed	The authorization code could not be written
isc_license_msg_dupid	The authorization code was not added to the license file because it is a duplicate of one already present in the file
isc_license_msg_convertfailed	The ID/key combination is invalid

TABLE 6.4 Error codes from *isc_license_add*()

isc_license_check()

Syntax

```
int isc_license_check(char *cert_id, char *cert_key)
```

Parameter	Type	Description
cert_id	char*	Pointer to a NULL-terminated character buffer containing the certificate ID to be checked
cert_key	char*	Pointer to a NULL-terminated character buffer containing the certificate key to be checked

Description Checks whether the specified ID/key pair is valid and could be added to **iblicense.dat**. Calling this function

Example To come after beta@@

Return Value *isc_license_check*() returns *isc_license_success* if it determines that the authorization code could be added. If it returns an error, pass the return value to *isc_license_get_msg*() to obtain the exact error message. The possible return values are:

Return	Description
isc_license_success	Authorization code could be successfully added
isc_license_msg_dupid	The authorization code was not added to the license file because it is a duplicate of one already present in the file
isc_license_msg_convertfailed	The ID/key combination is invalid

TABLE 6.5 Error codes from *isc_license_check*()

isc_license_remove()

Syntax

```
int isc_license_remove(char *cert_key)
```

Parameter	Type	Description
cert_key	char*	Pointer to a NULL-terminated character buffer containing the certificate key to be added

Description Removes the line specified by *cert_key* from **ib_license.dat**.

Example To come after beta@@

Return Value *isc_license_remove*() has the following return values:

Return	Description
isc_license_msg_restart	Authorization code was successfully removed
isc_license_msg_notremoved	The authorization code could not be removed; possible reasons are: • The key specified by *cert_key* does not exist in **ib_license.dat** • *cert_key* identifies an evaluation license

TABLE 6.6 Returns codes from *isc_license_remove*()

isc_license_display()

Syntax

```
unsigned short isc_license_display(char *buf, unsigned short buf_len)
```

Parameter	Type	Description
buf	*char**	• A character buffer for the result • Must be allocated by the programmer • *isc_license_get_message*() returns an error if *buf* is not long enough • Must be NULL-terminated
buf_len	short	• Length of *buf*

Description Places all certificate ID/key pairs that are currently in **iblicense.dat** into *buf*, separated by commas and NULL-terminated.

Example To come after beta@@

Return Value Returns zero if it succeeds. Otherwise, it returns the length that *buf* must have in order to contain the message text, and *buf* itself contains NULL.

isc_license_get_msg()

Syntax

```
unsigned short isc_get_msg(short msg_no, char *msg,
    unsigned short msg_len)
```

Parameter	Type	Description
msg_no	short	A message number returned by one of the other *isc_license_**() functions
msg	char*	• A character buffer for the message that corresponds to *msg_no* • Must be allocated by the programmer • Recommended length is 256 bytes
msg_len	short	The length of *msg*

Description When passed an error code from one of the other four functions in the License API, *isc_license_get_msg*() returns the text of the corresponding error message in the *msg* buffer.

Example To come after beta@@

Return Value *isc_license_get_msg*() returns zero if it succeeds. Otherwise, it returns the length that *msg* must have in order to contains the message text.

Pseudocode for a typical install

The following code indicates the steps you would typically take in writing an install. Calls to functions in the Install API and related specific code are in bold.

```
begin
    OPTIONS_HANDLE   handle;
    boolean          done = false;
    LANG_TYPE        language;

/* Get user preference if desired. This is if you created translated
 *  ibinstall.msg files in different directories */

language = get_language_choice();
if (language <> english)
    isc_install_load_external_text(lang_dirs[language]);

/* For an interactive install, check that OS is valid and Classic server
 * is not present before querying users for destination and options */

handle=0L;
isc_install_set_option(&handle, INTERBASE);
isc_install_precheck(handle, NULL, NULL);
isc_install_clear_options(&handle);

/* Query install for all the possible option names */

while(not all options)
    begin
        isc_install_get_info(isc_install_info_opname, option, opname buffer,
            ISC_INSTALL_MAX_MESSAGE_LEN);
        isc_install_get_info(isc_install_info_opdescription, option,
            opdesc buffer, ISC_INSTALL_MAX_MESSAGE_LEN);
```

```
        isc_install_get_info(isc_install_info_opspace, option, opspace buffer,
            sizeof(unsigned long));
    end;

/* Get a suggested destination directory */

isc_install_get_info(isc_install_info_destination, 0, dest_buffer,
    ISC_INSTALL_MAX_PATH);

/* Present the user his choices and interact with them */

interact_with_user();

/*  Use isc_install_set_option() and isc_install_unset_option() when
 *  interacting with the user or after the user clicks the Install button.
 *  Zero the handle before first call to isc_install_set_option(). /*

while (not all options)
    begin
        if(option is selected)
            isc_install_set_option(&handle, option); /* Check for errors.*/
    end;

/* You can check source_dir and dest_dir. */

error = isc_install_precheck(handle, source_path, dest_path)
    if (error > isc_install_success) then
        begin

/*  If a classic server is installed, or any server is running
 *  then give error and exit */

isc_install_get_message(error, message, length(message))
    user_choice = display(message);
        do_user_choice() /* Terminate, return to options selection screen */
            end
        else
            if (error < isc_install_success) then
              begin
```

```
/* Some warning has occured, display it and continue */

isc_install_get_message(error, message, length(message))
    display(message)
        end

display_file(install.txt)
display_file(license.txt)

/*  You are recommended but not required to supply callback functions
 *  because install aborts on any error. Some of the errors can be ignored.
 *  Some problems can be fixed by hand after the install. If you do not
 *  use callbacks you will not be able to inform the user of the status. /*

error = isc_install_execute(&handle, source_path, dest_path, NULL, NULL,
        NULL, NULL, NULL)
    if (error < 0) then
        begin
            isc_install_get_message(error, message, length(message))
                display(message)
                exit()
        end
        else
        if (error > 0) then
            begin
                isc_install_get_message(error, message, length(message))
                    display(message)
            end
display_file(readme.txt)

/* Clear options. not doing so results in memory leaks. */

isc_install_clear_options(&handle)
display_done()
end
```