Quintus Prolog Release 3.4

for Windows 95, 98, NT and 2000

Release Notes
and
Installation Guide

Swedish Institute of Computer Science (SICS)
PO Box 1263, SE-164 29 Kista, Sweden

Release Notes and Installation Guide, August 2000.

This manual corresponds to:

Quintus Prolog Release 3.4

SICS
PO Box 1263
SE-164 29 Kista, Sweden
+46 8 633 1500
http://www.sics.se/quintus/

Authorization codes:
Admin Department
qpadmin@sics.se

Problem reports and Product Suggestions:
Technical Support Group
qpsupport@sics.se

Additional product or service information:
Sales Department
qpsales@sics.se

Mailing list additions:
quintus-users-request@sics.se

RELEASE NOTES FOR PROLOG

R.1-1: Introduction

Quintus Prolog for the PC provides developers with the ability to create and deliver Quintus Prolog-based applications on x86 systems running Microsoft Windows 95, 98, NT and 2000. Quintus Prolog consists of both a Prolog Development System and Runtime Generator. These are based on Quintus Prolog Release 3.4, which Quintus supplies for the major UNIX-based platforms. Quintus Prolog is an embeddable implementation of Prolog, designed to support building software components that are easily integrated into applications written using other languages and tools.

Quintus Prolog for the PC features include:

- Interactive development system supporting incremental compilation of Prolog source code and featuring the Quintus 7-port Prolog debugger
- Runtime generator for building stand-alone Prolog executables. Windowed and console based applications can run under Windows 95, 98, NT and 2000.
- Full source code compatibility with Quintus Prolog 3.4 on UNIX-based platforms, with the exception of some platform-specific libraries
- Full use of the Quintus Object Format (QOF) files, which are portable across all supported platforms running Quintus Prolog 3.4
- Embeddability in C, C++ and Visual Basic
- The Quintus Prolog library, with the exception of some UNIX-specific libraries
- TCP library for interprocess communication support in Prolog applications
- Complete on-line manuals

R.1-2: About Prolog

Prolog is a high-level, general-purpose programming language based on predicate logic. Its key strengths are in the areas of reasoning, symbolic data manipulation, meta-programming and databases.

R.1-2-1: Quintus Prolog

Quintus Prolog adheres to the de-facto industrial Prolog standard, while providing extensive additional functionality including modules, stream I/O, libraries and support tools. Quintus Prolog's implementation emphasizes efficient memory management, very compact compiled code, high performance and comprehensive tools for application deployment.

One key advantage of Quintus Prolog is that it allows Prolog code to be embedded in programs written in other languages. Unlike most Prolog implementations Quintus Prolog has a very well-behaved discontiguous memory management system which does not require large amounts of space to be allocated in advance. This allows Quintus Prolog procedures to coexist with procedures from other languages such as C, C++ and Visual Basic.

Another important feature of Quintus Prolog is the Quintus Object Format (QOF) file. QOF files contain compiled Prolog code and are machine independent. This means that QOF files can be freely moved between any platform running Quintus Prolog 3.4.

For more information on these and other features of Quintus Prolog refer to Volume 1 *Language and Environment* of the Quintus Prolog manual set.

R.1-2-2: Quintus Prolog for the PC

Quintus Prolog for the PC includes the Development System and Runtime Generator that is fully compatible with the UNIX-based versions of Quintus Prolog 3.4. The Development System has the same functionality as the UNIX-based systems with the following exceptions:

- the profiler is not supported
- the Quintus User Interface, ProXL, ProXT and RPC libraries are not included
- the modules *aropen*, *nlist* and *menu* are not included in the main library

The runtime generator can be used with Microsoft Visual C++ (release 6.0 recommended). The executables can run under Windows 95, 98, NT and 2000.

Quintus Prolog provides full interoperability with Microsoft Visual C++. This means the Prolog procedures may call or be called by any C procedure giving complete embeddability with application programs.

R.1-2-3: What's New in Release 3.4

This section highlights the changes between this and Release 3.1 of Quintus Prolog.

From release 3.1 to 3.2:

- QP Window. A version of the development system is supplied that directs Prolog's standard streams to a Window. A new "-W" option to *qld* enables other Prolog applications to run with this window.

- Atom garbage collection, whereby memory for atoms no longer referenced by the Prolog execution can be reclaimed. With the introduction of this feature, the maximum length of atoms has also been increased from 1024 bytes to 65532 bytes. library(strings) has been modified to support the long atoms, only allocating space for long atoms as needed.
- The TCP library has been significantly enhanced to provide more efficient passing of terms between Prolog processes, more functionality for server processes and support for input callbacks, where predicates can be called either when a specified file descriptor becomes ready or at a specified time.
- A new licensing system has been introduced with this release. A license manager program is provided to setup the licenses and provide information on licenses installed.
- The cross-referencer tool has been replaced by a new version that is tuned to finding unreachable code in large programs. For details, refer to the `xref.doc` file in the tools directory.
- Finally, there are some minor changes in the directory structure with this release, in particular the host_type and system directory name has changed from *i386* to *ix86*.

From release 3.1 to 3.2:, the following bugs were fixed:

- `library(avl)`: procedure avl_fetch_1/5 is missing; misc. bugs
- `format/2` segmentation faults with large float exponents
- `set_input(user_output)` raises exception
- float unification in QOF files fails
- *qpc* syntax error message differs from development system
- `library(lists):nth1/3` description incorrect
- `use_module/2` information missing from QOF files
- `library(tcp)`: connection requests on callbacks functionality
- `library(date)`: `datime/7` format inconsistent with `time_stamp/3`
- `ord_list_to_avl/2`: creates avl tree with incorrect functor

From release 3.2 to 3.3:, the following bugs were fixed:

- Pascal Foreign Function Interface bug for `-string`
- bug in `library(putfile)`
- memory leak in `garbage_collect_atoms/0`
- stack shifter bug
- `atom_chars/2` with 256 element list bug
- *qpc* permanent variables limit detection
- *qpc* permanent variables limit detection

Release 3.4 features an Emacs-based source-linked debugger, with approximately the same functionality as the QUI-based debugger. See `manual(d-1-3)`.

The GNU Emacs interface was thoroughly revised and supports GNU Emacs 19 and 20.

It is now possible to build DLL:s containing Prolog code to be linked dynamically into applications (February 2000 patch release).

Quintus Prolog can be called from Visual Basic (February 2000 patch release).

From release 3.3 to 3.4, the following bugs were fixed, as well as a large number of documentation bugs:

- `current_predicate/2` failed after loading QOF-file saved by `save_predicates/2`
- re-exported predicates could fail if the `unknown` flag was `fail`
- dynamic code generation bug for imported predicates in runtimes
- bug in finding the correct module of an already loaded QOF file
- `user:message_hook/3` not obeyed on restore
- binary gets and puts broken
- CR/LF handling bug in the DOS console (February 2000 patch release)
- `functor/3` bug
- *qpc* bugs: input file suffix, `all_dynamic(true)` in embedded loads, handling `-N` and `-o` flags
- runtimes did not accept `Module:Head :- Body` clauses
- arithmetic bugs
- `retract/1` could leave a dangling pointer
- `ensure_loaded/1` did not work on QOF-files saved by `save_predicates/2`
- advice checking and debugging did not work when invoking a saved state
- DCG expansion of `{Goal}` bug
- C calling Prolog error handling bug
- re-exportation and meta-predicate interaction bug
- `save_modules/2` failed to save a dependency corresponding to `:- use_module(file,`[]).
- GC bugs
- more GC bugs (February 2000 patch release)
- indexing bug upon loading QOF-files
- prelinked QOF files were not considered up to date in binaries (February 2000 patch release)
- runtime statistics wrap-around bug
- handling of asynchronous input, EOF and interrupts in windowed executables such as `qpwin.exe`
- `QP_seek()` bug

- library bugs: `strings:span_*/*`, `strings:midstring/*`, `strings:concat_atom/*`, `strings:concat_chars/*`, `directory:file_member_of_directory/2`, `heaps:get_from_heap/4`, `date:time_stamp/*`
- `library(tcp)` bugs: floating point byte swapping, `tcp_shutdown()`, non-lingering `tcp_listen()`, file name size limitation

R.1-3: Requirements

- Microsoft Windows 95, 98, NT or 2000
- A 386, 486 or Pentium PC compatible machine
- At least 4 megabytes of available user memory
- At least 15 megabytes of available hard disk space
- Microsoft Visual C++

Microsoft Visual C++ is only required for the Runtime Generator and is not needed if you do not intend to build stand-alone Prolog executables.

R.1-4: Using Quintus Prolog

Quintus Prolog has two major components: the Development System and the Runtime Generator. The Development System is an interactive program for the incremental development of Prolog systems. There are two versions of the program: `prolog.exe` is a console-based program that can run in a DOS prompt window; `qpwin.exe` runs in its own window and directs the Prolog standard streams to that window.

The Runtime Generator includes a stand-alone Prolog compiler `qpc.exe` and QOF file linker `qld.exe` which, together with the Microsoft Visual C++ linker `link.exe`, can be used to build standalone Prolog executables to be delivered to end-users.

R.1-4-1: Starting the Development System

The batch file `qpvars.bat` included in the directory *quintus-directory*`\bin\ix86` can be executed to set up environment variables to run Quintus Prolog.

The console-based version can be run from a DOS prompt window as `prolog`. When Prolog starts up a copyright message is displayed followed by the Prolog prompt:

```
Quintus Prolog Release 3.4 (ix86, Windows NT)
Originally developed by Quintus Corporation, USA.
Copyright (C) 1998, Swedish Institute of Computer Science.  All rights re
PO Box 1263, SE-164 29  Kista, Sweden. +46 8 633 1500
Email: qpsupport@sics.se    WWW: http://www.sics.se/
Licensed to SICS

| ?-
```

The windowed version can be run from the File Manager or from the Program Manager if the `qpwin` executable is placed in an appropriate Program Group. This has the same appearance as the console-based version, except that the output is directed to a window. The properties of this window can be controlled by setting the environment variable `CONSOLE` to a comma separated list of:

`sl:`*INT*	history buffer size (default 200)
`cols:`*INT*	number of columns (default 80)
`rows:`*INT*	number of rows (default 24)
`x:`*INT*	X position in pixels
`y:`*INT*	Y position in pixels

For example, setting `CONSOLE` to `sl:400,rows:32` before starting `qpwin` yields a window with 32 rows and a history buffer of size 400.

When starting up, the Development System will look for the file `prolog.ini` in the directory specified by the environment variables `HOMEDRIVE` and `HOMEPATH`. These are defined automatically under Windows NT but have to be set manually in the other environments.

R.1-4-2: Creating Runtime Applications

Runtime executables may be built directly from source files and/or QOF files developed with Quintus Prolog version 3.4 on any platform. Both Prolog sources and machine-independent QOF files can be moved freely between different platforms, but foreign code will require recompilation on the target system.

To build an executable using the Runtime Generator, use `qld` to link together the application QOF files and any foreign object files or libraries. Here is a small example:

```
C:\> type hello.pl
runtime_entry(start) :- write(hello), nl.
```

The source file `hello.pl` can be compiled with *qpc* to a QOF file:

```
C:\> qpc -c hello.pl
```

Note that the compiler *qpc* can handle sources in both Windows and UNIX text file formats, so conversion between the two formats is not necessary. Then the *qld* command can be used to produce an executable from `hello.qof`:

```
C:\> qld -do hello.exe hello.qof
```

This will produce a console-based runtime executable named `hello.exe`.

R.1-5: Foreign Language Interface

With Quintus Prolog you can dynamically load foreign *Dynamic Link Libraries* or DLLs using **load_foreign_executable/1**, as well as foreign object files with **load_foreign_files/2**.

Foreign object dependencies will be correctly handled by *qld* so that the use of **load_foreign_files/2** to statically link in foreign code is fully supported.

To compile an object file `myprog.obj` to be loaded by **load_foreign_files/2**, use for example:

```
C:\> cl /c myprog.c
```

R.1-5-1: Dynamic Link Libraries (DLLs)

When using Microsoft Visual C++ to compile C source files for inclusion in a DLL, the `/MD` flag should be given to the C compiler `cl`. The `/dll` flag is given to the linker `link` to generate a DLL from object files. See the Microsoft Visual C++ documentation for more information on building DLLs. For example, the following command generates `myprog.dll`:

```
C:\> cl /MD myprog.c /Femyprog.dll /link /dll
```

If building a DLL that calls any of the Quintus Prolog C API functions exported from the Runtime Kernel DLL (`qpeng.dll`) or the Embedding Library DLL (`libqp.dll`) then you must also link in the import libraries for these DLLs, which are named `qpeng.lib` and `libqp.lib` and reside in the directory *quintus-directory*`\lib\ix86`. This is needed because DLLs must have their external references resolved at link time rather than at load time, in contrast to typical shared library implementations on UNIX.

The `makefile.win` file in the Quintus Library directory *quintus-directory*`\src\library` contains an example of how to build a DLL for dynamic loading into Quintus Prolog.

R.1-5-2: Building Windowed Executables

By default, a runtime executable will use console-based I/O. The **-W** switch to *qld* is provided to build windowed executables, which direct their I/O to a window.

```
C:\> qld -Wdo hello.exe hello.qof
```

This will produce a windowed runtime executable named `hello.exe`.

R.1-5-3: Building Static Executables

By default, a runtime executable will use the DLL version of the Runtime Kernel, as well as the DLL versions of any Prolog library foreign code. This will promote more sharing of code between different runtime applications. But delivering a dynamic runtime application can be difficult since all the DLLs must also be shipped along with the application in order for it to run.

Therefore, the **-S** switch to *qld* is provided to build a statically linked executable with no DLL dependencies. If static linking is used, the **-LD** switch must also be used, together with the library references `qpconsoles.lib user32.lib gdi32.lib comdlg32.lib`.

If the Prolog code has a foreign executable dependency on `myprog`, a static library `myprogs.lib` needs to be created. The trailing 's' is significant (see below); *qld* assumes this naming convention. Here is an example of the necessary sequence of commands to create a static executable `myprog.exe`:

```
C:\> cl /c myprog.c
C:\> lib /OUT:myprogs.lib myprog.obj
C:\> qpc -c myprog.pl
C:\> qld -Sdvo myprog.exe myprog.qof -LD \
     user32.lib comdlg32.lib qpconsoles.lib gdi32.lib
```

NOTE: `consoles.lib` of earlier releases has been renamed to `qpconsoles.lib` to avoid name conflicts.

The above command produces an executable which uses the static version of the Runtime Kernel and has no DLL dependencies. Statically linked applications can still dynamically load foreign code DLLs, provided these DLLs do not call any of the Quintus Prolog C API functions.

In order to distinguish static libraries from DLL import libraries in foreign dependencies the following naming convention has been chosen. If the **-S** flag is specified, when processing a library dependency *qld* will first search for the library with an 's' appended to its name, for example `libqps.lib` for the Embedding Library, and if not found it tries the original name.

The **-S** and **-W** flags can be combined.

R.1-5-4: Resource Files

qld recognizes files with the *.res* extension as resource files and treats them like object files, passing them to the linker. Resource files are generated from *.rc* files using the *rc* program supplied with Microsoft Visual C++. They contain various information relating to the Windows component of an application, such as the program name, icon and key bindings. The file `messages(system(qpwin.res))` contains such data for the Quintus Prolog window. Thus, the `qpwin.exe` executable can be generated with the command:

```
C:\> qld -WDdo qpwin.exe messages(system(qpwin.res))
```

The Quintus Prolog resource source file is in *quintus-directory*`\src\embed\qpwin.rc` and this references the icon file *quintus-directory*`\src\embed\qp.ico` which *rc* also incorporates into `qpwin.res`.

R.1-5-5: The *syslib* Path

The *syslib* file search path is provided to allow standard convention for Windows NT to be followed when searching for DLLs and libraries specified in **load_foreign_executable/1**. At startup time, Prolog asserts *syslib* file search path facts based upon the path specified in the environment variable `PATH` (as well as a couple of standard locations).

When running *qld*, the *syslib* file search path will be initialized to the path specified in the environment variable `LIB` in order to follow the Microsoft linker convention.

Therefore, the directive

```
:- load_foreign_executable(syslib(kernel32)).
```

executed in the Development System will load `kernel32.dll` from a directory in the `PATH` environment variable, whereas if it is encountered in *qld* the environment variable `LIB` will be used to locate the import library `kernel32.lib`.

The *syslib* file search paths can be modified by user code in the Development System or with the **-f** and **-F** options to *qld* if necessary.

R.1-6: Visual Basic Interface

Quintus Prolog can be called from Visual Basic. This interface is located in the directory *quintus-directory*`\src\vbqp` and comes with sources and a stand-alone manual; see `vbqp.pdf`. The implementation relies on calling Prolog code in a DLL, as described in the following section.

R.1-7: Building DLLs containing Prolog code

It is possible to build DLLs containing Prolog code to be linked dynamically into applications. Packaging your code as a DLL promotes sharing and is also a requirement for applications where an application needs to dynamically link your code. An example illustrating this and other techniques which are useful when embedding Prolog in C is the Visual Basic interface source code; see *quintus-directory*`\src\vbqp`.

Applications typically look for DLLs in the same directory as the application itself and also in any directories specified in the `PATH` environment variable. Applications also look in some system directories but that is less useful for our purposes.

The easiest way to ensure that the DLL (and the Quintus runtime DLLs) are found is to but the DLL together with the Quintus runtime DLLs in *quintus-directory*`\bin\ix86` and then ensure that the `PATH` environment variable is setup to include that directory, as described below.

Another method, especially suitable for running on a machine where Quintus Prolog is not installed, is to put all the DLLs in the same directory as the application. This includes the DLL built as below and the appropriate Quintus runtime DLLs.

See the Microsoft documentation for more information about DLLs.

To build a DLL containing Prolog code, follow these steps:

R.1-7-1: Setting up the environment

Set up your environment variables so that the Quintus binaries can be found. The batch file `qpvars.bat` included in the directory *quintus-directory*`\bin\ix86` can be run for this purpose.

R.1-7-2: Compiling the Prolog code

Compile your prolog code, e.g.

```
C:\> qpc -c vbqp.pl
```

The option `-c` ensures that only a Quintus object file (QOF) is produced, in this case `vbqp.qof`.

R.1-7-3: Compiling the C code

Compile your C code, e.g.,

```
C:\> cl /nologo /IC:\quintus\include /c /MD vbqp.c
```

The option `/MD` ensures that the DLL version of the C runtime libraries are used. The option `/c` ensures that no linking is performed, only an object file is produced, in this case `vbqp.obj`. The option `/I` tells *cl* where to find the `quintus.h` include file. It is redundant if `qpvars.bat` has been used to set up the environment.

R.1-7-4: Linking the DLL

Finally, use *qld* to link the DLL with the code produced in the previous steps and any additional libraries, e.g.,

```
C:\> qld -Y -o vbqp.dll -d vbqp.qof vbqp.obj -LD oleaut32.lib
```

NOTE: *qcon* will complain that there is no `runtime_entry/1`. This can be ignored.

The option `-o` is used to name the resulting DLL, in this case `vbqp.dll`. The option `-d` is used to ensure that any dependencies are also linked. The option `-LD` passes the rest of the command line to the linker, in this case it causes the library `oleaut32.lib` to be linked with the resulting DLL.

The option `-Y` (new in this release) tells *qld* to produce a DLL as opposed to an ordinary EXE file. A similar effect could be obtained with:

```
C:\> qld -o vbqp.dll -d vbqp.qof vbqp.obj -LD /dll oleaut32.lib
```

R.1-8: Sparse Memory Management Support

The low-level memory management support in Quintus Prolog for the PC is based on the Win32 Sparse Memory Management API. Consequently, Prolog applications running under Windows are more efficient in memory usage, as unused memory freed by the Prolog system will become globally available to other processes running on the system.

This is in contrast to implementations based on *malloc* and *free* as *free* will only return unused memory to the current process rather than the whole system.

R.1-9: Known Limitations

There are presently a few known limitations in Quintus Prolog for the PC that you should be aware of:

- In Windows 95/98, Control-C interrupt handling only works in windowed executables, and only when waiting for input.
- In statically linked executables running under Windows NT, Control-C interrupt handling only works when waiting for input.
- In a Windows NT console window, the command interpreter does not wait when a process executes an `execv()` library call. Thus when *qpc* calls *qld* it gives the appearance of running in the background. Similarly, when the **restore/1** predicate is executed.
- Redefining functions in the Embedding Library only works in dynamically linked executables.

INSTALLATION GUIDE

R.2-1: Introduction

This installation guide describes the installation of Quintus Prolog for the PC on Microsoft Windows 95, 98 and NT. Please read it carefully.

R.2-2: Installation Requirements

In order to use the Quintus Prolog Runtime Generator you must also install Microsoft Visual C++ (release 6.0 is recommended), although you are not required to install this prior to installing Quintus Prolog. The Microsoft Visual C++ compiler and linker tools must be found in your search path to use the Quintus Prolog Runtime Generator. Consult the Microsoft Visual C++ documentation for information on setting up your environment in order to do this.

R.2-3: Installing the Software

The installation of Quintus Prolog for the PC can be performed from a DOS prompt or a Windows NT console window. To install Quintus Prolog follow these steps:

- download the distribution anywhere e.g. into `c:\tmp`
- at the prompt, type **`c:\temp\install`** and press `<Enter>`
- follow the on-line instructions to complete the installation

Once the installation of Quintus Prolog is complete, a license must be installed with the program *qplm*. This is explained below.

The executables are installed in the subdirectory `bin\ix86` of the *quintus-directory* and this directory should be added to your file search path before using Quintus Prolog. Also installed is a batch file `qpvars.bat` which can do this for you as well as modifying the `LIB` and `INCLUDE` environment variables for using the Runtime Generator. You can either run this batch file manually or copy its contents to your `autoexec.bat` to setup environment variables for Quintus Prolog every time.

R.2-4: Installing a License

Having installed Quintus Prolog, the license file must then be set up with a license code in order to run *prolog*, *qpc*, etc. This is achieved with the license manager program *qplm*, which resides in the Quintus runtime directory (i.e. *quintus-directory*`\bin\ix86`).

The first step is to initialize the license file with *qplm -i <site name>*. The site name given to *qplm* here should be exactly the same as supplied with the code from Quintus since it is used to generate the code. For example:

```
C:\> qplm -i Hallatrow Designers Inc.
```

Then the next step is to add the products and codes. This is done with the command *qplm -a <product> <users> <code>*. The *<product>* argument is a string that comprises the product name ('prolog' in this case) the platform and the version number. The Windows 95, 98, NT and 2000 release thus requires a code for `prolog/ix86/3.4`.

The *<users>* argument specifies the number of regular users of Quintus Prolog and should usually be 1 on a PC; *<code>* is a 16 character string that is supplied by Quintus. To add a code for a 1 user license on the PC issue the command:

```
C:\> qplm -a prolog/ix86/3.4 1 thiscodewontwork
```

Quintus also provides temporary licenses that expire on a given date. The expiration date can be added with the command *qplm -a <product> <users> <expire> <code>*, where *<expire>* is an expiration date with the format YYYY-MM-DD. The following example adds a 1 user license that expires on 17 May 2004:

```
C:\> qplm -a prolog/ix86/3.4 1 2004-05-17 thisalsowontwork
```

You can print the information in the license file, including site name and products licensed with the *qplm -p* command.

R.2-5: Moving or Renaming the Quintus Directory

The installation program invokes `qsetpath.exe` to store path information such as the location of the Quintus directory into some of the installed executables and libraries. For this reason, if the location of the Quintus directory is changed, the path information must be updated for these executables and libraries in order to continue to run Quintus Prolog.

The installation program has an option **`-r`** can be used to do this after you have moved or renamed the Quintus directory. For example, if you want to move the contents of the Quintus directory from `C:\quintus` to `D:\quintus`, then you could enter the following commands at the prompt:

```
C:\> xcopy C:\quintus D:\quintus /s/e
C:\> D:\quintus\bin\ix86\install -r
```

This will copy the files and update the relevant Quintus Prolog files to the new location.

R.2-6: The GNU Emacs Interface

An interface between Quintus Prolog and GNU Emacs version 19 or 20 is provided. You can 'ftp' GNU Emacs from various sites free of charge, or buy it for media and duplication costs from:

```
Free Software Foundation          Voice:  +1-617-542-5942
59 Temple Place - Suite 330       Fax:    +1-617-542-2652
Boston, MA  02111-1307,  USA      gnu@gnu.org
http://www.fsf.org
```

Prolog will assume that GNU Emacs can be found as `emacs` in your file search path. Now to invoke *prolog* with GNU Emacs:

```
C:\> prolog +
```

After the Quintus Prolog banner appears, press the Control-x key followed by the Control-c key (^X^C) to exit.

More details can be found in `manual(d)`.

RELEASE NOTES AND INSTALLATION GUIDE