IP Toolbench v1.2.11
Developer's Guide

Author(s)
Yogesh Gathoo, Bradley Grove
IP Business Unit

Revision History

Date	Version	Comment
02/14/2002	1.0	Initial Draft
02/21/2002	1.1	After first level of feedback
04/17/2002	2.0	Adding support for 1.0.1
04/29/2002	2.1	Some grammatical / spelling fixes
05/08/2002	2.2	Report plug-in removed/ minor other additions
08/19/2002	2.3	IPTB1.1 Additions
10/10/2002	2.4	Perl API added
10/16/2002	2.5	Appendix "Encryption Details" added
10/31/2002	2.6	Super Legacy/ Enhanced Legacy Plugin
11/01/2002	2.7	Updated Command Arguments & Super Legacy plug-in
11/07/2002	2.8	Update to Super Legacy Plugin
12/20/2002	2.9	Changes to simplify PTF
01/06/2003	3.0	Major restructuring corresponding to simplification and improving readability.
01/14/2003	3.1	Proofreading edits
01/15/2003	3.2	Added contents for MEGACORE & Creating new plug ins
4/11/2003	3.3	Updated the default of the Java_dir command line option and added configuration information for the default Perl libraries.
5/20/2003	3.4	Includes wizard.ptf samples for various IP Toolbench versions. Look at TOC
8/18/2003	3.5	IP Toolbench 1.2 kick off
11/17/2003	4.0	1.2.2 Revisions
11/17/2003	4.1	Updated based on feedback from Yogesh
11/18/2003	4.2	Update to LegacyMode section based on input from Dave Moore.
11/24/2003	4.3	Added info on –devicefamily / -projectname / -projectdir switches.
12/10/2003	4.4	Enhancements to Simgen Plugin
1/6/2004	4.5	Added Simgen "parameter" setting
1/20/2004	4.6	Added section_name attribute to DefaultDocumentationPlugin and other minor edits

4/23/2004	4.7	Changing core data in Non-MVC wizard section added
9/27/04	4.8	Minor updates for v1.2.7, including support for Tcl scripts in defailtAddProjectFiles.
7/12/05	4.9	Update for Message Window support added to CustomCoreWizard Plugin
11/3/05	4.10	Updates for 1.2.11, including design assistant warning suppression
8/8/2006	5.0	First updates for 1.3.0 – HDLGenerator Plugin

Table of Contents

Purpose of this Document

This document is intended for IP Developers seeking to understand how to use IP Toolbench v1.2.11 with their IP core releases. It covers various aspects of customization and guidelines for packaging and deployment.

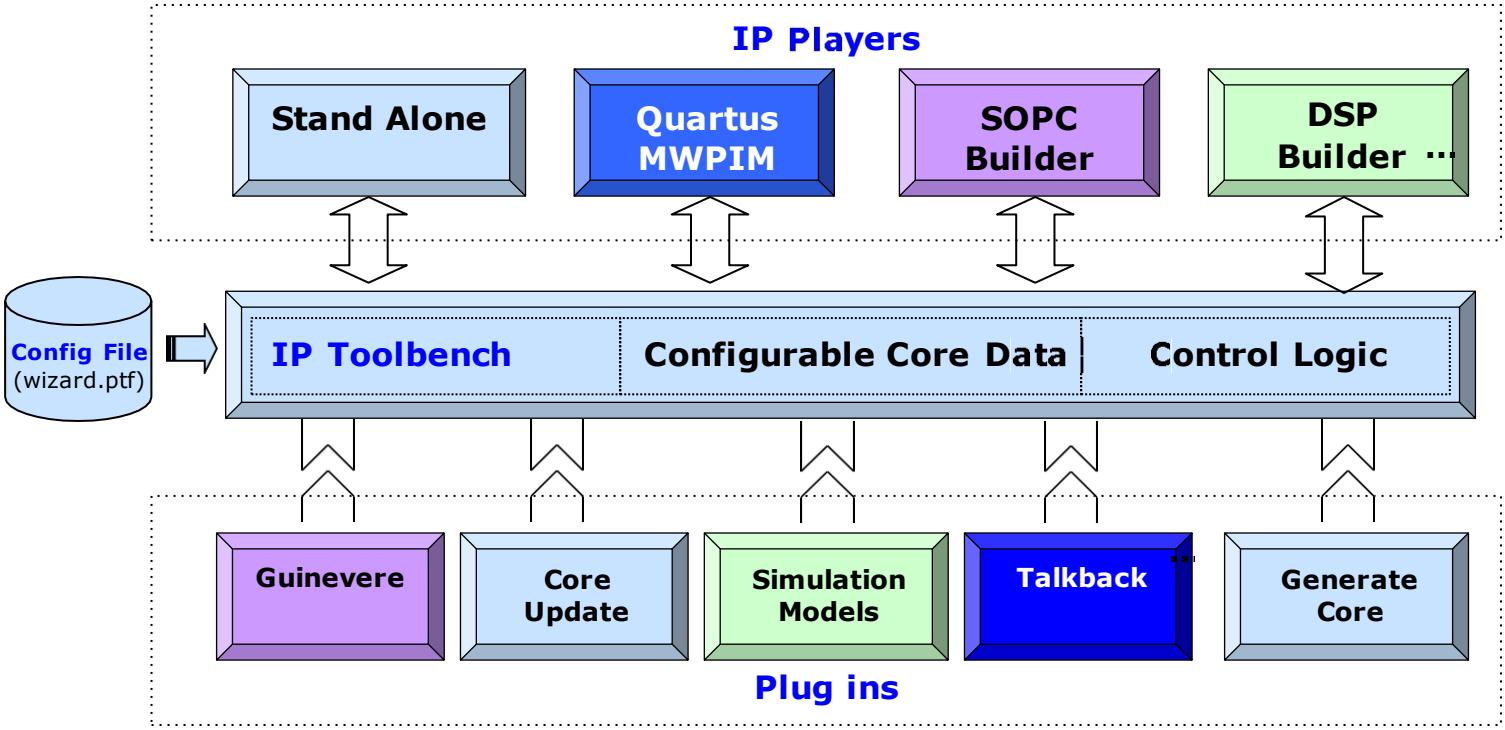
Conventions

- Blue text is used to indicate code or code related synonyms
- All PTF sections are in caps
- All PTF attributes are in lower case.
- All caps heading indicate PTF section where appropriate

IP Toolbench Overview

IP Toolbench is a framework designed for developing MegaWizards. It has a component architecture that enables IP Developers to select, customize and configure "plug-ins" to implement the IP customization process on the customer's desktop. The framework holds the core configuration data in a global database that can be read by any plug-in.

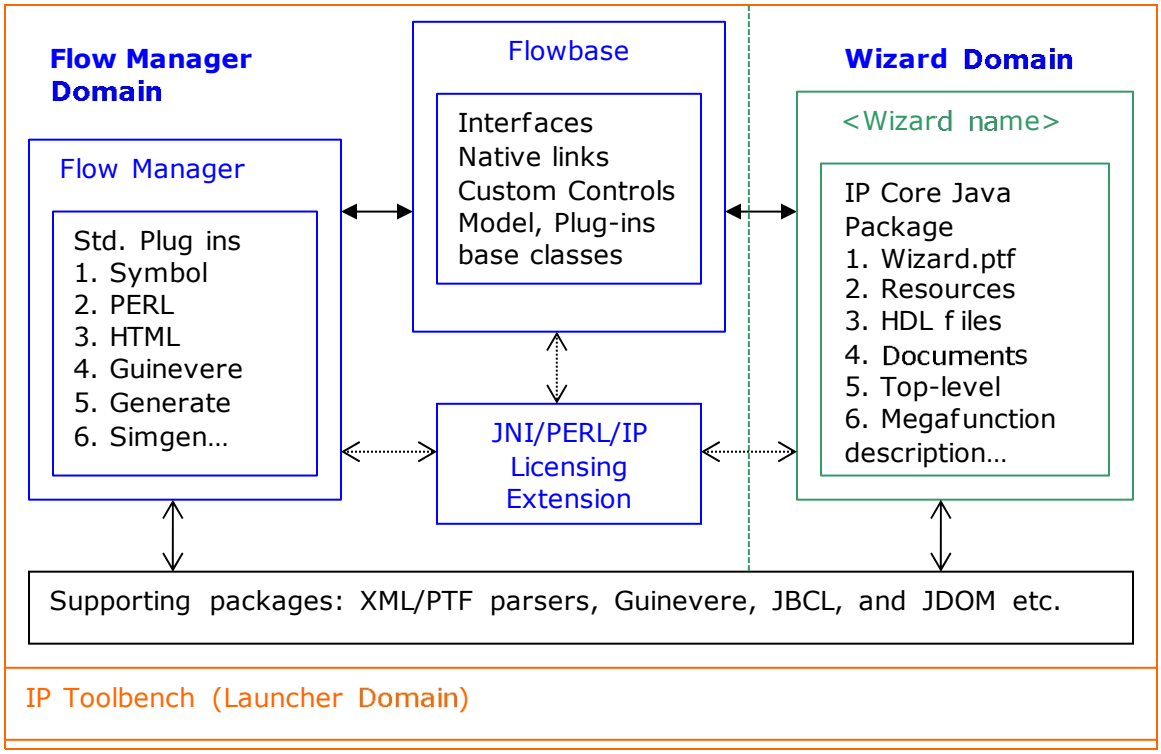
The following diagram illustrates the IP Toolbench component architecture. Each plug-in performs a task: "Guinevere" displays the parameterization GUI, "Core Update" checks for a new version of the core on Altera's website, and "Generate Core" creates a top-level wrapper that parameterizes the core.



The IP Toolbench framework is built on central configuration file (wizard.ptf) that defines the core, initializes IP Toolbench configurable core data, and specifies which plug-ins should be loaded for this core. Currently, this configuration file is written in PTF format, but we plan on eventually migrating this to XML using Corespec.

The IP Toolbench package contains six modules:

- Flowbase: Java interfaces and base classes for IP Toolbench framework and plug-ins. Flowmanager: Implements IP Toolbench framework as defined in Flowbase package.
- Launcher: Entry point to IP Toolbench, implements Framework Class Loader that loads IP Toolbench with supporting packages and launches it. This allows us to implement a deferred linking of components.
- <Wizard name>: Jar package created by the IP Developer, contains wizard.ptf (IP Toobench configuration File) and core specific resources, possibly including Java and Perl code. Refer to the section "What does the <Wizard name>.jar file contain?" for more info.
- <Supporting Packages>: Additional third party packages needed to run IP Toolbench or the wizard specific code. The packages include in the IP Toolbench distribution are:
    - JDOM - http://www.jdom.org - Distributed under the JDOM license http://dev.eclipse.org/viewcvs/indextech.cgi/org.eclipse.stellation/downloads/libs/about-jdom.html?rev=HEAD
    - XERCES - http://xml.apache.org/ - Distributed under the Apache License - http://xml.apache.org/dist/LICENSE.txt
    - Perl - http://www.perl.org/ - Distributed under the Artistic License - http://www.perl.com/lpt/a/language/misc/Artistic. html
    - STLPort - http://www.stlport.org/ - Distributed under the STLPort License - http://www.stlport.org/doc/license.html
- ePerl: A native C++ based library linked into IP Toolbench. This library includes the Altera IP encryption and licensing libraries, along with an embedded Perl v5.8.3 interpreter.



What does the <Wizard Name>.JAR file contain?

- **Configuration Files**
  1. Master Configuration File (wizard.ptf)
  2. Guinevere GUI description (which may also be included as part of wizard.ptf)
- **Java / Perl Code**
  1. Model or Top-level generator to create Ports and Constants
  2. Any other core/design flow specific plug-ins
- **Manifest.MF**
  1. Wizard package version though the Manifest-Version attribute
  2. Location of the master configuration file through the Wizard-Database attribute

  The following is an example of a wizard package Manifest.MF file:
  Manifest-Version: v1.2.0
  Wizard-Database: altera/ipbu/WDivideWizard/wizard.ptf

A Jar file is a Java archive file. It is an aggregation of files, similar to a zip or tar file. It is created using the jar utility included with the Sun JDK. Before creating the Jar package, we put all of the files into a subdirectory named "**altera/ipbu/<Wizard name>**". We then package the entire tree into the Jar file. This directory structure ensures that the core specific files are in a unique Java package.

The IP Developer is free to include other files into the <wizard name>.jar files. Some examples of other files that might be included are:

- **Documentation**
- **HTML Pages**
- **HDL / Design Files**

What does the IP Toolbench Daily Build contain?

Nightly builds are created by checking out the latest code off the main branch of the CVS tree. IP Developers can access these builds using through the Windows share located at \\sj-ipbusoft\builds or the the NFS mount to the apps2 partition (/apps2/IP/builds). Each version of IP Toolbench has a separate build directory in this share. Within each version directory are the numbered nightly builds. These builds remain for 2 weeks after creation before being deleted. In addition, releases builds are marked with a "rel_" prefix. Release builds will never be deleted. Each version directory has a directory named "latest" that contains the last successful build for that version.

IP Toolbench v1.2.7 supports Windows NT/2000/XP, Red Hat Linux 7.3/8.0/Enterprise 3 WS, and Solaris 8 and 9.

An IP Toolbench build includes:

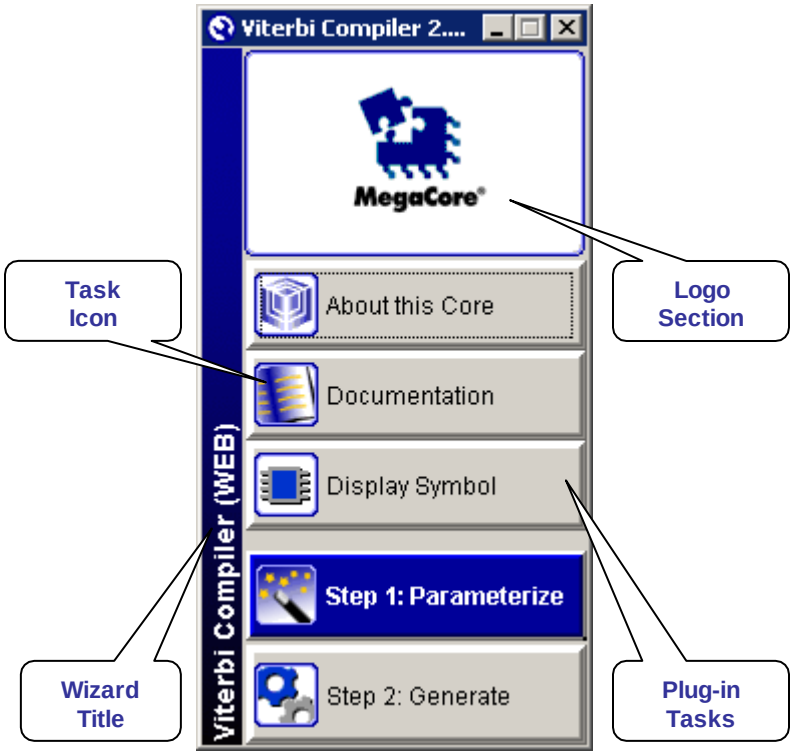
- **IP Toolbench Tarball:** A .tar.gz file containing the IP Toolbench build.
- **JRE:** Java Runtime Environment 1.4.1. While IP Toolbench v1.2.2 and above will use the JRE inside of the Quartus bin directory by default, a Java runtime is also included in the build to allow us to run the IP Toolbench executable on machines where a valid Quartus installation is not available. The Java Runtime is not included in the tarball.
- **Devtools:** Contains
  1. A build of Perl.exe supporting Altera's 3DES IP Encryption. This is a standalone Perl 5.8.3 interpreter used for developing scripts to be included with IP Toolbench. Inside IP Toolbench, we intend for these scripts to run using the Perl plug-in.

2. Perlcrypt.exe for encrypting Perl scripts and HDL files using 3DES encryption. Perlcrypt encrypted files cannot be read by Quartus, only by the IP Toolbench Perl extensions specifically intended to read Perlcrypt files.
3. eperl.dll, Native library working with perl.exe
4. Javadocs for IP Toolbench framework

- **ip_toolbench:** Contains a versioned sub folder with:
  1. ip_toolbench.exe: Executable to launch IP Toolbench
  2. flowbase.jar
  3. flowmanager.jar
  4. eperl and other dependencies
  5. util directory containing Altera developed and third party packages such as xerces.jar, sopc_wizard.jar, jdom.jar, netlist_to_hdl.jar. These packages are required by IP Toolbench.

IP Toolbench Screen Shot

- Screen shot for v1.2.x and earlier



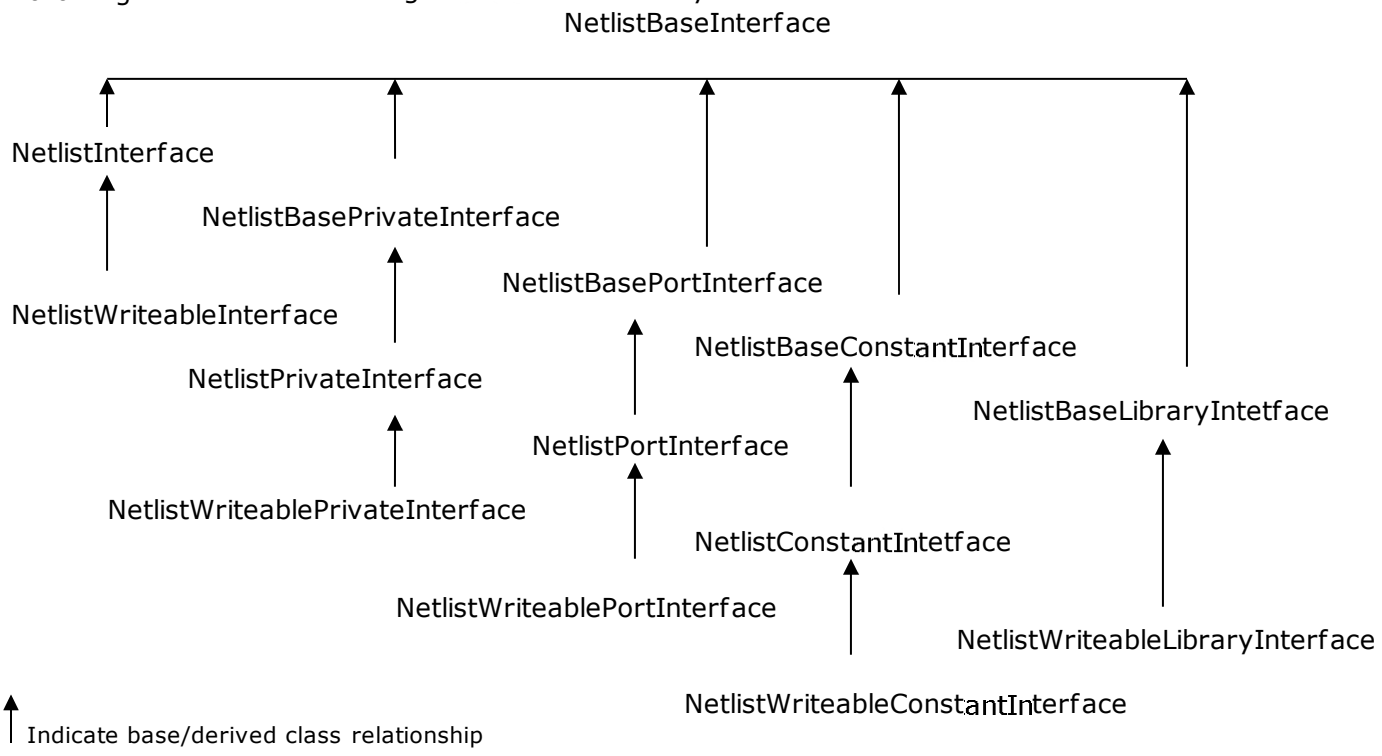
Netlist Class Hierarchy

IP Toolbench is a framework for writing plug-ins that act on the configurable core data. The plug-ins view of this data is the Netlist Class Hierarchy. A hierarchical list of these classes follows:

ModelBaseClass encapsulates all Netlist objects and offer MVC based base implementation

- **NetlistInterface** composing context IP Core module with its parameters, ports and constants
  - **NetlistPortInterface** contains a list of ports and methods to add, remove, and modify them.
    - **NetlistPort** class encapsulating individual port
  - **NetlistConstantInterface** contains a list of constants, which correspond to VHDL generics and Verilog parameters, and methods for adding, removing and modifying them.
    - **NetlistConstant** class encapsulating each individual constant
  - **NetlistFileInterface** contains a list of HDL files used to implement this IP core and methods for adding, removing and modifying files.
    - **NetlistConstant** class encapsulating each individual file.
  - **NetlistBaseLibraryInterface** contains a list of VHDL libraries and methods for adding, removing and modifying them.
    - **NetlistLibrary** class encapsulating each individual library
  - **NetlistGlobalPrivateInterface** contains a list of global privates and namespaces based on plug-ins
    - **NetlistPrivateInterface** contains a list of privates for each namespace and methods to add, remove or modify them.
      - **NetlistPrivate** class encapsulating each individual private at the namespace level
        - **Range** describes valid value set for a private
    - **NetlistPrivate** class encapsulating each individual private at the base level

Following is an UML describing the Netlist hierarchy



Refer to the IP Toolbench javadocs for more detailed description on methods and data members of these classes.

Changing core data in NON-MVC Wizards

For changing core module data in NON-MVC wizards developers will have to typecast NETLIST(), PORTS(), CONSTANTS(), PRIVATES() and FILES() to respective writeable interfaces

For example, to change the module name in a NON-MVC wizard, typecast NETLIST() object to NetlistWriteableInterface and call setModuleName() method on it. Refer to IP Toolbench Java Docs for more information.

```
((NetlistWriteableInterface)NETLIST()).setModuleName("sl_top");
```

Configuring IP Toolbench for your wizard

The most critical part of defining an IP Toolbench wizard for your core is the creation of the IP Toolbench configuration file. This section explains the format of the configuration file in detail.

Designers can control, configure IP Toolbench framework through a single file named "wizard.ptf.' This file contains PTF structured information organized under single root section called "MEGACORE"

MEGACORE

Data in wizard.ptf file is organized under a root section named MEGACORE. MEGACORE contains four sections:

1. A description of core: title, version, ordering codes and other details.
2. A NETLIST_SECTION that holds port and parameter data for the core.
3. A SOPC_BUILDER section that defines how this core interfaces to an SOPC Builder system using Avalon. This section is optional. It is only required if the IP Toolbench will be launched under SOPC Builder.
4. A PLUGIN_SECTION that lists the IP Toolbench plug-ins used to configure this core.

Table of Attributes

Name	*Type*	*Description*	*Default*
title	String	Full "longname" of the megafunction.	NULL
version	String	Version of the megafunction release. Must be prefixed with letter 'v'	NULL
format_version	String	Specifies the wizard.ptf version. For IP Toolbench v1.2.x, this must be set to 120.	NULL
short_title	String	Short name for your megafunction. This should be less than 17 characters.	NULL
release_date*	String	Date when core was released	NULL
product_id*	String	Unique ID for your megafunction. For Altera MegaCores, this is assigned by IPBU marketing	NULL
vendor*	String	Your company name	NULL
vendor_id*	String	Your company ID assigned by the IPBU marketing group.	NULL

copy_rights*	String	The copyright message	NULL
opencoreplus*	String	"Yes" if supported else "No"	NULL

*Attributes used by "About this Core" html page

The Megacore section contains sub-sections for listing ordering codes, supported devices, certifications, and simulation models. Each of these sections contains attributes that are treaded as a list. Each of these sections is used by the "About this Core" page.

```
ORDERING_CODES
{
  code = "IP-WDIVIDE";
}
SIMULATION_MODELS
{
  support = "IP Functional Simulation Model (VHDL / Verilog HDL(";
}
CERTIFICATIONS
{
  certificate = "SOPC Builder Ready";
  certificate = "Atlantic(TM) Compliant";
}
DEVICES
{
  device = "Stratix(TM)";
  device = "Stratix GX";
  device = "Stratix II";
  device = "Cyclone(TM)";
}
```

IP Developers can add more information to the MegaCore section and link it with their HTML pages. Refer to Java HTML Viewer Plug-in for more information.

The simplest valid Wizard.ptf looks as follows:

```
MEGACORE
{
#Attributes for title and version
        title = "Viterbi Megacore Megafunction";
        version = "v1.0.0";
        short_title = "Viterbi";
        … #About this Core related stuff
        #Megacore netlist related info
        NETLIST_SECTION
        {…}

        #IP Toolbench plug-ins info
        PLUGIN_SECTION
        {…}
} #end of MEGACORE section
```

NETLIST_SECTION

The NETLIST_SECTION contains a section named STATIC_SECTION.

Any data in the NETLIST_SECTION can be dynamically generated. Typically, the wizard will dynamically generate the ports and constants.