Installation and usage of isystem.connect

Introduction

This document contains instructions for *isystem.connect* for all supported languages. Common steps are described in the first section, language specific instructions are given next. The following languages are currently supported:

- C++
- Python
- Java
- C#
- Perl (deprecated)
- Tcl (deprecated)
- Matlab

General instructions

Documentation

Documentaion for each SDK is stored in folder *documentation*. Each SDK contains this file and *isystem.connect* API documentation, which is generated from C++ header files. If you installed SDKs with winIDEA, documentation is already unzipped and you can open it with winIDEA menu option `Help | SDK`. If you downloaded SDK from iSystem's web page, you have to unzip documentation zip file and open file *index.html* in your web browser. Save its location to browser bookmarks for future reference. Other files in folder *documentation* are SDK specific. If they are *zip* files, unzip them and open file *index.html* in your web browser.

Installers and libraries

SDKs also contain installers or libraries, which we need to use *isystem.connect* from other languages. The exact instructions are given for each language below.
If you'd like to know the version number of the already installed module, print string returned by function `getModuleVersion()`. See instructions for each language for an example.

Examples

Each SDK has examples, which demonstrate usage of *isystem.connect* and *testIDEA*. All examples are based on winIDEA project stored in folder *targetProjects*. This folder contains sources, winIDEA workspace files (*.xjrf), and object and download files. The sources enable you to change and compile projects, but if you don't have compiler available, you can still try scripts and testIDEA with bundled download files. If you'd like to build this project for your target, edit project settings in winIDEA. There is also flag `USE_FLOAT_TYPE` in *projectDefs.h*, which you can comment if you don't have floating point libraries available for your controller.
Additional change you may need to make is changing stack addresses in testIDEA *File | Properties | Stack usage* or deactivate stack measurements.

List of target example projects:
- Workspace *Sample5554.xjrf* contains project for MPC5554.
- Workspace *SampleSTM32.xjrf* contains project for STM32.
- Workspace *SampleSimulator5554.xjrf* contains project, where winIDEA is configured to run simulator (MPC5554). The simulator supports only the following functionality:

- debug without batch access
- isystem.test, if it does not use analyzer (trace, profiler, coverage)
- jtag scan (the returned data has no meaning)

The list of samples working with simulator is specified for each language below.

Folder *itest*

This folder contains support files for *isystem.test* API and *testIDEA*:
- templates for Python and Perl script generation
- XML Style sheet and CSS files for test reports
- XML Schemas for test reports

Which language?

If you can't decide about which language to use for isystem.connect, this section may help you. Usually the following criteria is important:

Which language do we already know and use?
Many people decide accoding to this answer. It enables a quick start, but may not be the best in long term. Don't be afraid of learning new computer languages.

What do we want to implement?
If we only want to write short scripts for testing and development, scripting language

is preferred, because it offers the highest productivity. If we intend to write full blown IDE on top of *isystem.connect*, then some of compiled, static languages would be better.

For scripting languages we recommend **Python**. It is integrated into winIDEA, has very readable syntax, and has many good libraries. For compiled languages we recommend **Java**. It is used by Eclipse so also our Debug plug-in for Eclipse is implemented in Java. Most development tools for Java are free and it also has many good libraries.

Language Modules

C++

Documentation

This folder contains C++ API documentation and *isystem-exceptions-api-doc.zip*. The latter describes exceptions, which are thrown by *isystem.connect* C++ methods. You may also use them in your programs.

Installation

To use *isystem.connect* from C++ application, we have to add libraries from *lib* folder to our project and include header files from *include* folder.
To see version of the installed *isystem.connect* module, execute the following code:

```
#include "CUtil.h"
cout << "Version of isystem.connect: " << isys::getModuleVersion() << '\n';
```

Examples

Example in *examples* folder is build with Visual Studio 2008 SP1. Express edition with SP1 can also be used.

Python

Documentation

This folder contains two additional files:

- *Python-and-winIDEA.pdf* - this file contains instructions on how to use Python from winIDEA.
- *isystem-connect-python-doc.zip* - this file contains documentation for *isystem.connect* utilities for Python and instructions for using *isystem.connect* from Python. If SDK was installed with winIDEA setup, this file is already unzipepd.

Please note that *isystem.connect* module for Python also contains docstrings, so API documentation is at hand.

Installation

To add support for *isystem.connect* to Python, we should run the installer in *installer* folder, and follow the installation wizard.
To see version of the installed *isystem.connect* module, execute the following code:

```
import isystem.connect as ic
print ic.getModuleVersion()
```

Examples

Folder *examples* contains several example scripts. They can be run from winIDEA or from command line. The following examples work with simulator:

- chartSample.py
- itestSample.py, except the last test case in the script, which uses coverage
- jtagSample.py
- readTargetVariable.py
- testSpecVsTestControllerSample.py

For other examples we have to modify the winIDEA workspace to work with our target.

Java

Documentation

This folder contains API documentation.

Installation

To create your own project, copy files from *lib* folder to your Java project, and add the code, which loads the *dll* to your code. See *examples/IConnectJavaSample*, method *main()* for example on how to load the *dll*.
Add also the jar file from *lib* folder to your classpath.
To see version of the installed *isystem.connect* module, execute the following code:

```
System.out.println("Version of isystem.connect: " +
                   si.isystem.connect.connect.getModuleVersion());
```

Examples

Folder *examples* contains Eclipse project, which you can add to Eclipse workspace and build and run it.

jdoc and IConnectJavaSource.zip

jdoc folder contains API documentation generated with Javadoc tool. Please note that this documentation is autmatically translated
from C++ code comments in doxygen format. Since Doxygen supports more tags than Javadoc, translatied result is not 100%
equal to the original - especially some formatting may be lost. Therefore it is recommended to use API doc in the documentation
folder as a primary source. However, this documentation can be used when seeking for help from IDE, for example Eclipse.

IConnectJavaSource.zip file contains zipped sources with code comments, which can be used by Eclipse.
Eclipse can show Javadoc API and method comments in *Help* view and a tootip-like window, but we must configure it to find the
documentation. This can be acheived by right clicking the project in Package Explorer view and setting Java Build Path, Libraries, IConnectJNI.jar, properties *Source attachment* and *Javadoc location* as shown in the image below:

Set values for 'Source attachment' and 'Javadoc location' for IConnectJNI.jar in build path.

Source attachment should point to *IConnectJavaSource.zip* file from SDK, and *Javadoc location* should point
to *jdoc* folder from SDK.

C#

Documentation

This folder contains API documentation.

Installation

To use *isystem.connect* from C# application, you have to add libraries from *lib* folder to the project. Add .NET assembly *IConnectCSLib.dll* to *References* section of your C# project.

It is very important to use the right *isystemConnect.dll* from SDK *lib* folder: either from *lib\win32* or *lib\x64*. Depending on your target platform, you must copy the right *isystemConnect.dll* to your project or other folder on Windows DLL search path. The **BadImageFormatException** is most likely indication of wrong *isystemConnect.dll* version.

To see version of the installed *isystem.connect* module, execute the following code:

```
    Console.WriteLine("\nVersion of isystem.connect: " +
isystemConnect.getModuleVersion());
```

Examples

Example in *examples* folder is build with Visual Studio 2008.

Perl (deprecated)

SDK for Perl is no longer actively developed. Contact iSystem support for further information.

Documentation

This folder contains API documentation.

Installation

To use *isystem.connect* from Perl, module files from SDK's *lib-x.y* folder should be accessible, where *x.y* identifies Perl version. Modules are built for Activestate's Perl 5.8.8, 5.8.9, 5.10, 5.12.3, 5.16.3, and RTRT 5.8. Usually we should copy them to *lib*

directory in Perl's installation folder.
To see version of the installed *isystem.connect* module, execute the following code:

```
use isystemConnect;
print isystemConnect::getModuleVersion()
```

Examples

Folder *examples* contains several example scripts. The following examples work with simulator:

- `readTargetVariable.pl`
- `jtagSample.pl`
- `itestSample.pl`, except the last test case in the script, which uses coverage

For other examples we have to modify the winIDEA workspace for our target.

Tcl (deprecated)

SDK for Tcl is no longer actively developed. Contact iSystem support for further information.

Packages are built for Activestate's Tcl 8.5.8 and 8.6.0. Please note, that Tcl wrapper of *isystem.connect* provides low level interface and Object Oriented interface. It is possible to mix both of them, but it is not recommended. SDK contains examples with OO interface.
More information is available at

http://www.swig.org/Doc1.3/SWIGDocumentation.html#Tcl_nn21

Different explanation (for the older versions of SWIG, but may also be useful to read):

http://www.swig.org/Doc1.1/HTML/Tcl.html#n3

Documentation

This folder contains API documentation.

Installation

To use *isystem.connect* from Tcl, copy folder *isystemConnect* from folder *lib* to folder searched by Tcl for packages, for example *Tcl\lib\tcl8.5*. Execute the following line to find out the Tcl search path:

```
% puts $tcl_library
```

The final disk contents should be (assuming Tcl in installed to *C:\Tcl*):

```
C:\Tcl\lib\tcl8.5\isystemConnect\isystemConnect.dll
C:\Tcl\lib\tcl8.5\isystemConnect\pkgIndex.tcl
```

To see version of the installed *isystem.connect* module, execute the following code in *tclsh*:

```
package require isystemconnect
puts "[getModuleVersion]"
```

If you'd like to copy *isystem.connect* package to some other directory, you can use the following code to load the package:

```
lappend auto_path "my/folder/with/isystemconnect/here"

package require isystemconnect

...
```

Examples

Folder *examples* contains several example scripts. The following examples work with simulator:

- readTargetVariable.pl
- jtagSample.pl
- itestSample.pl, except the last test case in the script, which uses coverage

For other examples we have to modify the winIDEA workspace for our target.

Matlab

isystem.connect for Matlab is based on *isystem.connect* for Java. The API is therefore the same, but there are few exceptions:

- libraries can not be loaded directly with *java.lang.System.loadLibrary()* call. Use the provided *ICLoader* class from `'ISystemConnectLoader.jar'` instead. See also examples.

- enumeration constants can not be accessed directly as in Java, but you have to use 'javaMethod' call instead. Example:
  `IConnectDebug.EAccessFlags.fMonitor` can be accessed with:
  `fMonitorEnum = javaMethod('valueOf',`
  `'si.isystem.connect.IConnectDebug$EAccessFlags', 'fMonitor');`

Documentation

Folder *documentation* contains API documentation. Unzip it and bookmark file *index.html* in your web browser.

Installation

Unzip the SDK file, then copy the *lib* folder to your preferred location. Then you should add *ISystemConnectLoader.jar* and *IConnectJNI.jar* to Matlab's static or dynamic Java classpath path and load the library. This procedure is described below, but it is also implemented in the example script `icInit.m`, which is part of the SDK, while details are available online.

Loading the library

Before using isystem.coonect API, we must load the library.
java.lang.System.loadLibrary() call may not be executed from Matlab window, but we should use *si.isystem.connect.mutil.ICLoader* instead. Example:

```
javaaddpath('..\lib\ISystemConnectLoader.jar');
javaaddpath('..\lib\IConnectJNI.jar');
loader = si.isystem.connect.mutil.ICLoader;
```

Now we can load native libraries from paths stored in `'java.library.path'`. It is important to use '/' as separator here. Current directory ('.') should be in `'java.library.path'`, and we must NOT specify the extension. Example:

```
loader.loadLibraryFromSysPath('../lib/IConnectJNI');
```

Another possibility is loading from absolute path. In this case we must also specify extension 'dll', for example:

```
loader.loadLibraryAbs('d:\lib\IConnectJNI.dll');
```

Setting of `'java.library.path'` in Matlab is described here. We can get the current value in Matlab with:

```
java.lang.System.getProperty('java.library.path')
```

Please note that *IConnecJNI.dll* and *IConnectJNI.jar* files are the same as for Java, so we may keep only one copy of these two libraries on the system. See also the example script `icInit.m` from the SDK.

Examples

Folder *examples* contains m-files, which we can run and use as startup for our projects. This folder also contains the file `librarypath.txt`, which sets the `java.library.path` system variable, if we start Matlab in this directory. Otherwise we have to add the current directory to `librarypath.txt`, or modify the file `icInit.m` to load the *isystem.connect* native library `IConnectJNI.dll` from absolute path.

To run examples, first run the script `icInit.m` to load required libraries. Then you can run other scripts.

Typical script

The first step in using *isystem.connect* is establishing connection to winIDEA:

```
cMgr = si.isystem.connect.ConnectionMgr();
cMgr.connectMRU('');
```

Then we instantiate other classes, depending on operations we want to perform. For example, if we want to execute debug operations (reset, run, runUntilFunction, ...), we instantiate class *CDebugFacade*:

```
debug = si.isystem.connect.CDebugFacade(cMgr);
```

Now we can call methods:

```
debug.download();
debug.runUntilFunction('main');
debug.waitUntilStopped();
```

Documentation of all classes and their methods is available at iSystem web page.

Simulink

There are no general Simulink blocks provided with *isystem.connect* calls, but you can use Matlab function block and call *isystem.connect* methods from there.