Icetips Utilities

Classes and Templates

10/14/2018

Icetips Utilities

Published: October 2018

Publisher

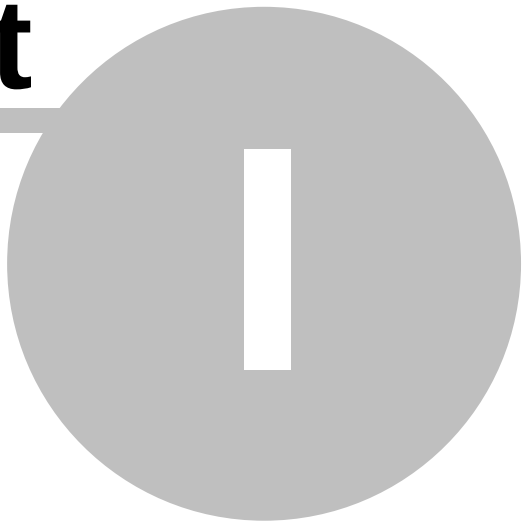
Icetips Creative, Inc.

Managing Editor

Arnor Baldvinsson

Table of Contents

Part



Chapter 1 - Icetips Utilities

1 Icetips Utilities

Welcome to the Icetips Utilities build 2018.10.2460.323

Documents updated	Sunday, October 14, 2018
Version/Build	2018.10.2460.323
Build Date	October 14, 2018
Copyright	Copyright ©2007-2018 Icetips Alta LLC.
Author	Arnór Baldvinsson

This documentation is based on the classes in release dated October 14, 2018. Note that some documented classes have not had all the properties and methods documented yet.

The Icetips utilities 2018.10.2460.323 build contains the following:
42 Classes
577 Methods
243 Properties
32 Extension templates
6 Utility templates
4 Code templates
2 Control templates

This help file details each property and method in each class separately with as much detail as we can. The documentation is installed into "%ROOT%\3rdParty\Docs\Icetips Utility Class\ITUtility.chm" where %ROOT% is the Clarion Root directory depending on which version of Clarion you choose to install the Classes for.

To add the Icetips Utility Class to your application, add the"Icetips Utility Classes Global 451" global template to your applications. It must be applied to all applications that use the classes. The classes are ABC compliant and use a named "ITUTIL" group as definition:

```
!ABCIncludeFile(ITUTIL)
```

The classes are compatible with Clarion 6, Clarion 7 and newer. They may be compatible with Clarion 5.5H but they are **not compatible with Clarion 5.5G or earlier**. We only test with Clarion 6.3 and the latest builds of Clarion 10. Note that support for Clarion 6 will end by the end of 2015. We will test with Clarion 6 until the end of 2014.

Unless otherwise noted, classes should be instantiated at procedure level. There should be no threading considerations in Clarion 6 and above as far as we know since all instances are at procedure level. The only exception is the Global window threading class, which use critical sections and should be thread safe!

Start Here 5

Thank you for using our Icetips Utilities build 2018.10.2460.323, released on Sunday, October 14, 2018. This is a set of Clarion classes and templates that add various functionality to your programs.

This help file details each property and method in each class separately with as much detail as we

can. Some of the classes may be undocumented when this document is released so please bear with us if something is missing. We have spent 4 years writing these classes as our needs demanded and the task of documenting is monumental since we did not do that as we wrote the classes.

Completed documentation

Classes

Global extension templates

Procedure Extension templates

Code Templates

Utility Templates

We are constantly adding new functionality to these classes and templates. If you have any suggestions for new classes, methods or properties, please let us know.

Tutorial Videos

Progress Class

Progress Class Tutorial Video - Part 1 Length: 15 min, 13 sec.
Progress Class Tutorial Video - Part 2 Length: 21 min, 29 sec.

1.1 Start Here

Welcome to the Icetips Utilities templates. This documentation is an on going project and may not correctly show all options available in the current release.

Documents updated	Sunday, October 14, 2018
Version/Build	2018.10.2460.323
Build Date	October 14, 2018
Copyright	Copyright ©2007-2018 Icetips Alta LLC.
Author	Arnór Baldvinsson

Unless you are using a hand coded project, you need to apply the Global Extension template 451 to ALL applications where you are using the classes. If you are using Legacy applications you need to add the "Icetips Utilities Classes Global - LEGACY ONLY" Global Extension template to your application(s) For all intents and purposes it is identical to the ABC Global Extension template 451.

If you are using hand coded projects, please check out the chapter about using Icetips Utilities in hand coded projects 8.

In multi-dll systems the Global Extension template 451 must also be added to the root/data dll and the root dll app must be re-compiled to export the classes for any dependent applications.

How to add the classes to your application

To add the Icetips Utilities to your application use the "Icetips Utility Classes Global 451" global template to your Global Extensions. This gives you access to all the classes in that file in your application. In multi-dll systems you must apply this to the exporting (root) dll and then to any and all applications where you are going to use the Utilities and then rebuild the system starting with the exporting/root dll.

How to use the classes in your procedures

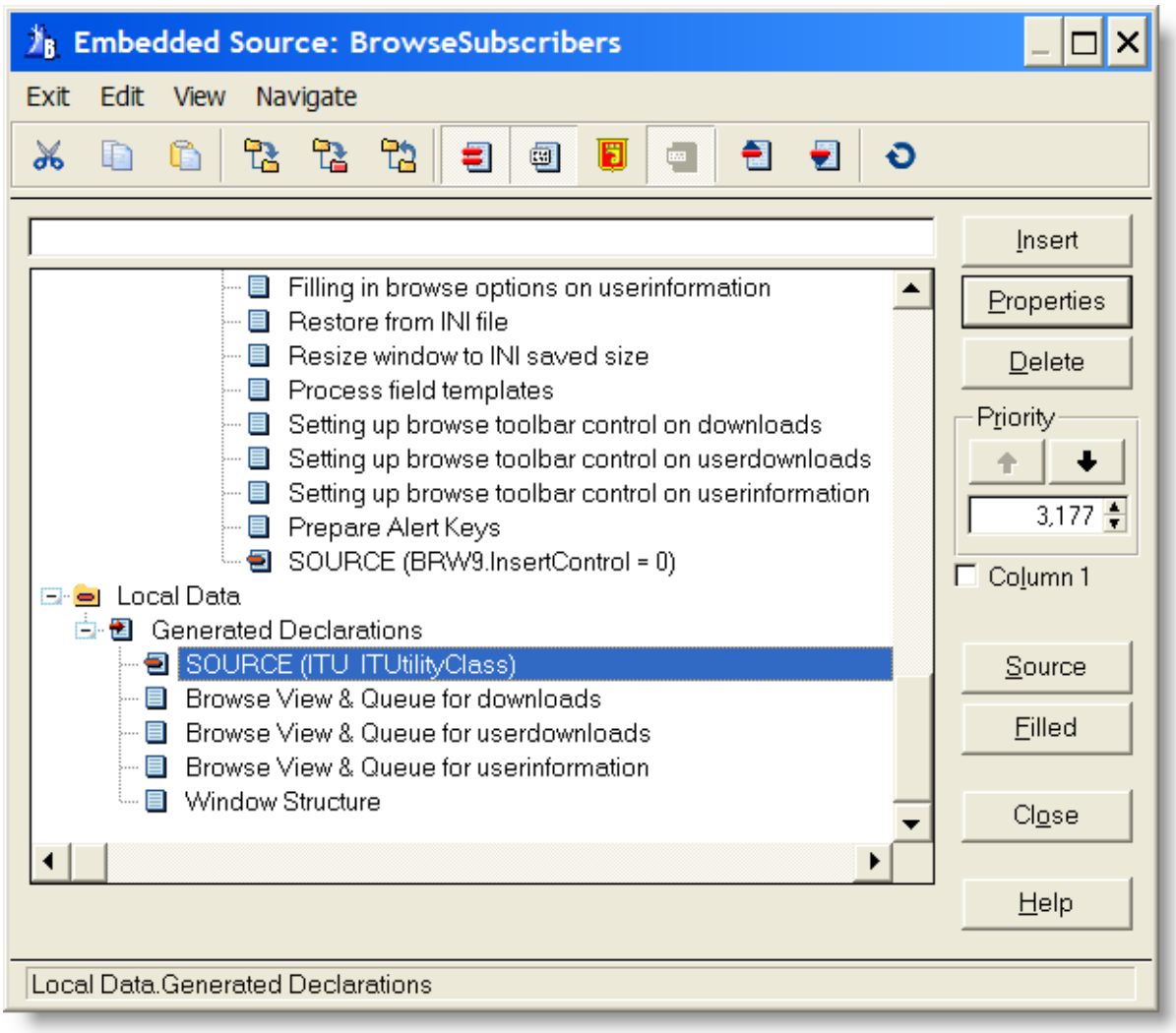
To instantiate a class in your procedure you simply declare an instance. Go to the Local Data embed and add the instantiation code, such as:

```
ITU  ITUtilityClass
```

or:

```
ITS  ITStringClass
```

etc. You can derive the classes, but most of the methods are very specific so you generally don't need to override or derive. Some classes may need deriving to override virtual methods. The screenshot below shows how the ITUtiltyClass is instantiated in a procedure.

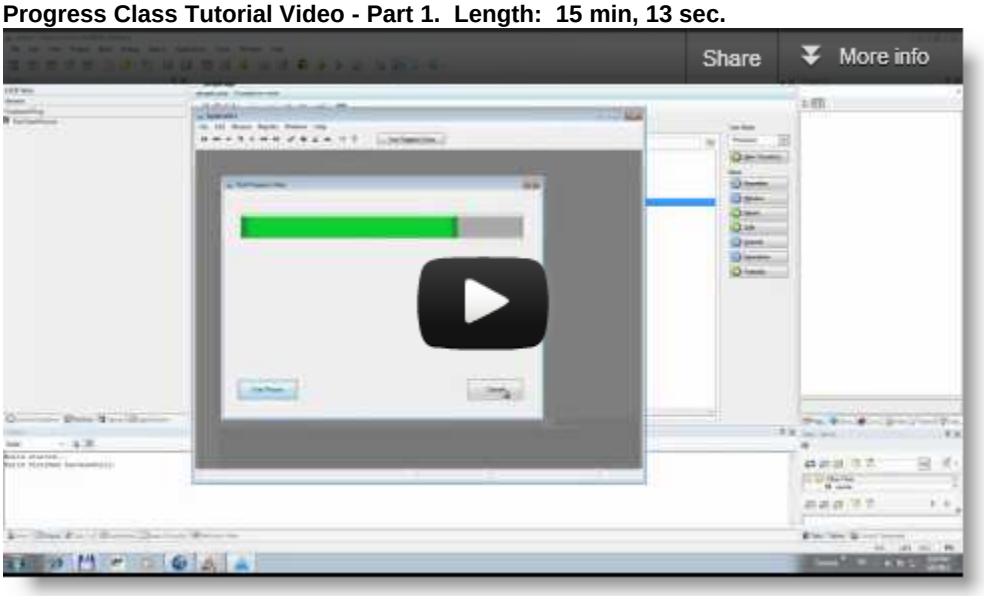


Once you have instantiated the class you can access the methods in it.

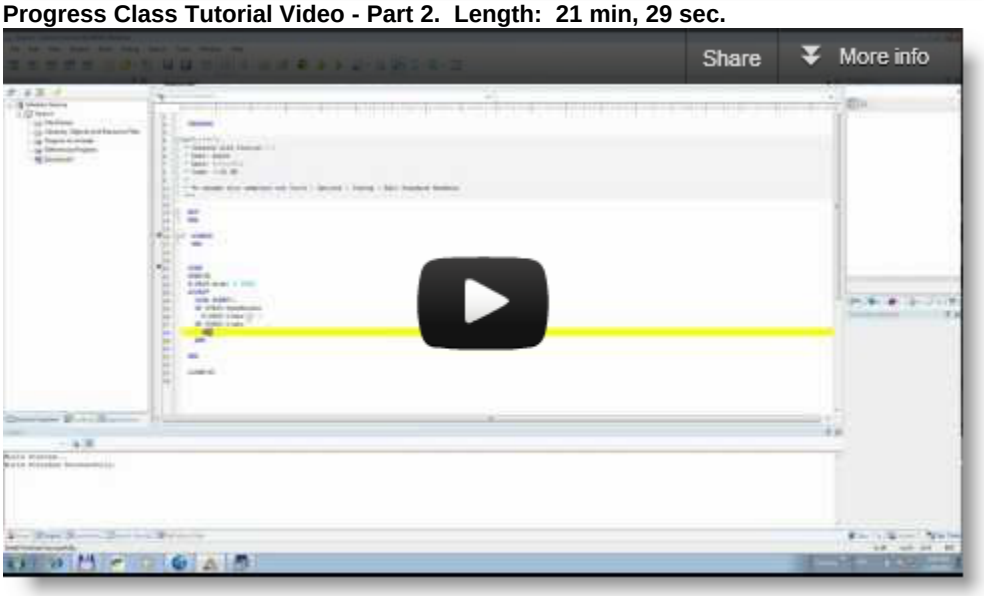
1.2 Tutorial Videos

In May 2012 we started creating tutorial videos on Youtube for our products. As videos become available we will add links to them on the corresponding topic pages in the documentation and they will also be listed here. You can always go to http://www.youtube.com/user/IcetipsVideos/videos to see what videos are available. We also post information about them on our blog.

Progress Class Tutorial Video - Part 1. Length: 15 min, 13 sec.



Progress Class Tutorial Video - Part 2. Length: 21 min, 29 sec.



1.3 Hand coded projects

Please see this blog post http://www.icetips.com/blog_wp/2015/06/16/using-icetips-utilities-in-hand-coded-projects/ about how to use the Icetips Utilities in hand coded projects in Clarion 10.

The section below applies to Clarion 6.

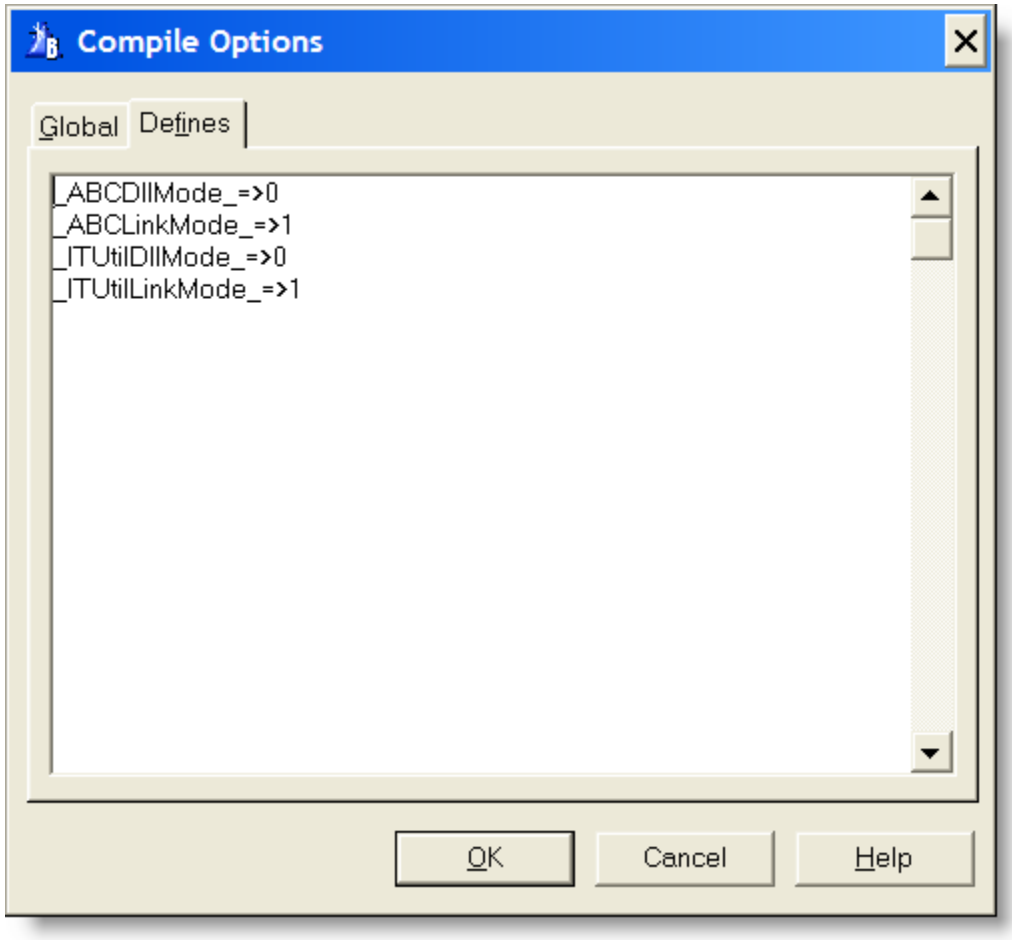
To use the **Icetips Utilities classes** in hand coded projects you need to make slight modification to the defines in your project by adding the following to the project defines:

```
_ABCDllMode_=>0
_ABCLinkMode_=>1
_ITUtilDllMode_=>0
_ITUtilLinkMode_=>1
```

Or add this to your .prj file:

```
#pragma define(_ABCDllMode_=>0)
#pragma define(_ABCLinkMode_=>1)
#pragma define(_ITUtilDllMode_=>0)
#pragma define(_ITUtilLinkMode_=>1)
```

before the #compile statement. See screenshot below.



If you are working with multi-dll hand coded project, this setting is appropriate for the exporting dll. All dependent dlls and exes must reverse the values, i.e.

```
_ABCDllMode_=>1
_ABCLinkMode_=>0
_ITUtilDllMode_=>1
_ITUtilLinkMode_=>0
```

You also need to include the class source into your code by adding:

```
Include('ITUtilityClass.inc'),ONCE
```

after your MAP/END statement.

1.4 Compile issues in Clarion

Up to and including version 6.3 build 9056 there were no issues with compiles in Clarion. However in 9057 and 9058 certain problems were introduced.

Version 6.3, build 9057 introduced options to use OMIT() in ABC header files, but it was not implemented correctly and this caused problems with some of our classes. Build 9058 fixed this.

Version 6.3, build 9058 had a problem with the locally linked runtime library causing duplicate symbols in certain DLL apis. The fix is to use the local runtime library from 9057! Please refer to http://www.cwaddons.com/company/errata.html where you can download zip files with lib files for the win32 library and winInet library.

As of January 2016, the Core class cannot be used by itself in **Legacy** Applications without causing a GPF. Use any other class instead as they all inherit from the Core Class but none of them cause a GPF. This ONLY applies to Legacy (Clarion) applications, not ABC applications.

1.5 Documentation Conventions

We have tried our best to produce a uniform and standardized documentation for our classes.

Please note that properties and methods are sorted alphabetically in this documentation index. However that is not always exactly the same as in the ITUtilities.inc file. In the source the Construct and Destruct method are always at the end of the method list. This may not always be in the documentation although we try to remember to put them at the end.

1.6 Coding conventions

I follow pretty strict coding convention that makes the code look good and hopefully very readable to our users. People use different styles for writing code and I have mine. Since i also work with case sensitive languages I try very hard to make sure that things are always kept in the same kind of capitalization so it's not too difficult for me to jump from one project to the next!

The rules are simple:

1. Two (2) character indents. This has not been enforced in some of the products that we have taken over from other developers but we have been working on it.
2. CODE statement is in column 2. There are exceptions to this rule where it is in column 1. Following code should be the same indent as the CODE statement.
3. Mixed case statements, i.e. KeyCode() instead of keycode() or KEYCODE()
4. Upper case logical keywords, i.e. NOT, AND, OR
5. Only full upper case keywords are SELF and PARENT
6. Never, EVER use period (.) instead of End, ever! There are, however, exceptions to this in some of the product code that we have taken over. I never use the period in my code! The exceptions are always single lines, like:

   ```
   If SELF.GetHdr(HeaderID) = False Then Return .
   ```

7. Space between statements and arithmetic characters, such as X + 1 instead of X+1 or S & X instead of S&X
8. Clarion properties are upper cased PROP then mixed case, i.e. PROP:LineHeight
9. Constructors and Destructors are declared after other methods in both the declaration (.inc) and implementation (.clw) files.
10. Class properties are declared before methods. There may be exceptions to this during development when isolating specific methods and properties but in product builds this rule should always be followed.
11. Comments that indicate updates/fixes generally always start with TWO exclamation marks (!!) initials of the developers (AB) and the date in YYYY-MM-DD order, like:

    ```
    !! AB 2012-06-25:  Problem with calculation fixed.
    ```

12. Every method is identified with a !== comment line above and !-- comment line below. Above the !== line should be two empty lines to separate methods from one another.

1.7 License Agreement

Icetips "Utilities" End-User License Agreement

Important - read carefully!

By installing this software you have agreed to be bound by the following End-User Licence Agreement.

ICETIPS ALTA LLC ("ICETIPS") IS WILLING TO LICENSE THE SOFTWARE ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS SOFTWARE LICENSE AGREEMENT. PLEASE READ THE TERMS CAREFULLY. BY CLICKING ON "YES, ACCEPT" OR BY INSTALLING THE SOFTWARE, YOU WILL INDICATE YOUR AGREEMENT WITH THEM. IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF A COMPANY OR OTHER LEGAL ENTITY, YOUR ACCEPTANCE REPRESENTS THAT YOU HAVE THE AUTHORITY TO BIND SUCH ENTITY TO THESE TERMS, IN WHICH CASE "YOU" OR "YOUR" SHALL REFER TO YOUR ENTITY. IF YOU DO NOT AGREE WITH THESE TERMS, OR IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ENTITY, THEN ICETIPS IS UNWILLING TO LICENSE THE SOFTWARE, AND YOU SHOULD SELECT THE "NO, DECLINE" BUTTON AND THE DOWNLOAD OR INSTALL WILL NOT CONTINUE.

SOFTWARE LICENSE AGREEMENT

1. Parties. The parties to this Agreement are you, the licensee ("You") and Icetips. If You are not acting on behalf of Yourself as an individual, then "You" means Your company or organization.

2. The Software. The Software licensed under this Agreement consists of computer programs only in compiled, object code form, data compilation(s), source code and documentation referred to as Icetips subscription product (the "Software").

3. Subscription Term For Registered User Version. The term of the license granted herein for the registered version of the Software shall be on a subscription basis with an initial term of one (1) year, and optional recurring renewal terms of one (1) year each, unless prior to renewal this license is terminated by written notice by You for convenience or terminated by either party for material breach. Renewal procedures are described in the accompanying documentation, and unless such procedures are strictly satisfied, including the payment of any required license fee, Your updates to the Software is not authorized, but use of Your existing Software is authorized. No updates or upgrades to the Software can be authorized unless the license fee is paid.

4. Registered Version License Grant for Single Copies (Non-Network Use). If You are a registered user of the Software, You are granted non-exclusive rights to install and use the Software by a single person who uses the Software only on one or more computers or workstations. You may copy the

Software for archival purposes, provided that any copy must contain the original Software's proprietary notices in unaltered form.

5. Registered Version License Grant For Network Use. If You are a registered user of the Software, You are granted non-exclusive rights to install and use the Software and/or transmit the Software over an internal computer network, provided You acquire and dedicate a licensed copy of the Software for each user who may access the Software concurrently with any other user. You may copy the Software for archival purposes, provided that any copy must contain the original Software's proprietary notices in unaltered form.

6. Restrictions. You may not: (i) permit others to use the Software, except as expressly provided above for authorized network use; (ii) modify or translate the Software; (iii) reverse engineer, decompile, or disassemble the Software, except to the extent this restriction is expressly prohibited by applicable law; (iv) create derivative works based on the Software; (v) merge the Software with another product; (vi) copy the Software, except as expressly provided above; or (vii) remove or obscure any proprietary rights notices or labels on the Software.

7. Purchase of Additional Licenses. Registered users of the Software may purchase license rights for additional authorized use of the Software in accordance with Icetips' then-current volume pricing schedule. Such additional licenses shall be governed by the terms and conditions hereof. You agree that, absent Icetips' express written acceptance thereof, the terms and conditions contained in any purchase order or other document issued by You to Icetips for the purchase of additional licenses, shall not be binding on Icetips to the extent that such terms and conditions are additional to or inconsistent with those contained in this Agreement.

8. Transfers. You may make a one-time permanent transfer of all of your license rights to the Software to another party, provided that all of the following conditions are satisfied: (a) you notify us in writing of your intent to transfer your license rights and identify the party receiving the Software with complete contact information; (b) the transfer must include all of the Software, including all its component parts, original media, printed materials and this License Agreement; (c) you do not retain any copies of any version of the Software, full or partial, including copies stored on a computer or other storage device; and (d) the party receiving the Software reads and agrees to accept the terms and conditions of this License Agreement. Notwithstanding the foregoing, we reserve the right to require the transfer of possession of all physical copies of the Software to us for purposes of re-issue of replacement copies to the party receiving the Software.

9. Ownership. Icetips and its suppliers own the Software, all physical copies thereof, and all intellectual property rights embodied therein, including copyrights and valuable trade secrets embodied in the Software's design and coding methodology. The Software is protected by United States copyright laws and international treaty provisions. This Agreement provides You only a limited use license, and no ownership of any intellectual property. We reserve the right to require

you to transfer possession of all physical copies of the Software to us for purposes of re-issue of replacement copies.

10. Warranty Disclaimer; Limitation of Liability. ICETIPS PROVIDES THE SOFTWARE "AS-IS" AND PROVIDED WITH ALL FAULTS. NEITHER ICETIPS NOR ANY OF ITS SUPPLIERS OR RESELLERS MAKES ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. ICETIPS AND ITS SUPPLIERS SPECIFICALLY DISCLAIM THE IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SYSTEM INTEGRATION, AND DATA ACCURACY. THERE IS NO WARRANTY OR GUARANTEE THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED, ERROR-FREE, OR VIRUS-FREE, OR THAT THE SOFTWARE WILL MEET ANY PARTICULAR CRITERIA OF PERFORMANCE, QUALITY, ACCURACY, PURPOSE, OR NEED. YOU ASSUME THE ENTIRE RISK OF SELECTION, INSTALLATION, AND USE OF THE SOFTWARE. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS AGREEMENT. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

11. Local Law. If implied warranties may not be disclaimed under applicable law, then ANY IMPLIED WARRANTIES ARE LIMITED IN DURATION TO THE PERIOD REQUIRED BY APPLICABLE LAW. Some jurisdictions do not allow limitations on how long an implied warranty may last, so the above limitations may not apply to You. This warranty gives you specific rights, and You may have other rights which vary from jurisdiction to jurisdiction.

12. Limitation of Liability. INDEPENDENT OF THE FORGOING PROVISIONS, IN NO EVENT AND UNDER NO LEGAL THEORY, INCLUDING WITHOUT LIMITATION, TORT, CONTRACT, OR STRICT PRODUCTS LIABILITY, SHALL ICETIPS OR ANY OF ITS SUPPLIERS BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER MALFUNCTION, OR ANY OTHER KIND OF COMMERCIAL DAMAGE, EVEN IF ICETIPS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT PROHIBITED BY APPLICABLE LAW. IN NO EVENT SHALL ICETIPS'S LIABILITY FOR DAMAGES FOR ANY CAUSE WHATSOEVER, AND REGARDLESS OF THE FORM OF ACTION, EXCEED IN THE AGGREGATE THE AMOUNT OF THE PURCHASE PRICE PAID FOR THE SOFTWARE LICENSE.

13. Export Controls. You agree to comply with all export laws and restrictions and regulations of the United States or foreign agencies or authorities, and not to export or re-export the Software or any direct product thereof in violation of any such restrictions, laws or regulations, or without all necessary approvals. As applicable, each party shall obtain and bear all expenses relating to any necessary licenses and/or exemptions with respect to its own export of the Software from the U.S. Neither the Software nor the underlying information or technology may be electronically transmitted or otherwise exported or re-exported (i) into Cuba, Iran, Iraq, Libya, North Korea, Sudan,

Syria or any other country subject to U.S. trade sanctions covering the Software, to individuals or entities controlled by such countries, or to nationals or residents of such countries other than nationals who are lawfully admitted permanent residents of countries not subject to such sanctions; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals and Blocked Persons or the U.S. Commerce Department's Table of Denial Orders. By downloading or using the Software, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

14. U.S. Government End-Users. The Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Software with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished rights are reserved under the copyright laws of the United States.

15. Licensee Outside The U.S. If You are located outside the U.S., then the following provisions shall apply: (i) Les parties aux presentes confirment leur volonte que cette convention de meme que tous les documents y compris tout avis qui siy rattache, soient rediges en langue anglaise (translation: "The parties confirm that this Agreement and all related documentation is and will be in the English language."); and (ii) You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software, and You represent that You have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

16. Severability. If any provision of this Agreement is declared invalid or unenforceable, such provision shall be deemed modified to the extent necessary and possible to render it valid and enforceable. In any event, the unenforceability or invalidity of any provision shall not affect any other provision of this Agreement, and this Agreement shall continue in full force and effect, and be construed and enforced, as if such provision had not been included, or had been modified as above provided, as the case may be.

17. Arbitration. Except for actions to protect intellectual property rights and to enforce an arbitrator ’s decision hereunder, all disputes, controversies, or claims arising out of or relating to this Agreement or a breach thereof shall be submitted to and finally resolved by arbitration under the rules of the American Arbitration Association ("AAA") then in effect. There shall be one arbitrator, and such arbitrator shall be chosen by mutual agreement of the parties in accordance with AAA rules. The arbitration shall take place in Port Angeles, Washington, USA, and may be conducted by telephone or online. The arbitrator shall apply the laws of the State of Washington, USA to all issues in dispute. The controversy or claim shall be arbitrated on an individual basis, and shall not be

consolidated in any arbitration with any claim or controversy of any other party. The findings of the arbitrator shall be final and binding on the parties, and may be entered in any court of competent jurisdiction for enforcement. Enforcements of any award or judgment shall be governed by the United Nations Convention on the Recognition and Enforcement of Foreign Arbitral Awards. Should either party file an action contrary to this provision, the other party may recover attorney's fees and costs up to $1000.00.

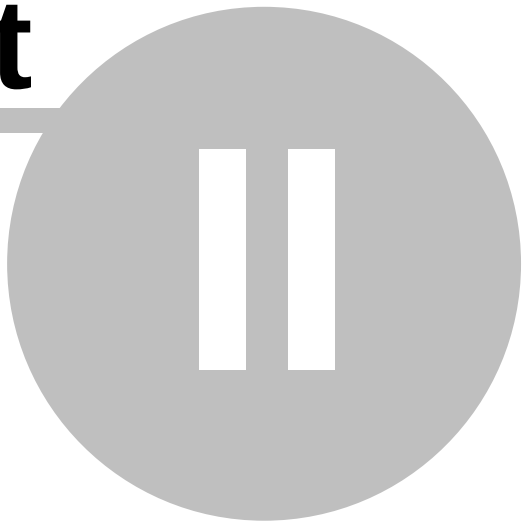
18. Jurisdiction And Venue. The courts of Clallam County in the State of Washington, USA and the nearest U.S. District Court shall be the exclusive jurisdiction and venue for all legal proceedings that are not arbitrated under this Agreement.

19. Force Majeure. Neither party shall be liable for damages for any delay or failure of delivery arising out of causes beyond their reasonable control and without their fault or negligence, including, but not limited to, Acts of God, acts of civil or military authority, fires, riots, wars, embargoes, Internet disruptions, hacker attacks, or communications failures. Notwithstanding anything to the contrary contained herein, if either party is unable to perform hereunder for a period of thirty (30) consecutive days, then the other party may terminate this Agreement immediately without liability by ten (10) days written notice to the other.

20. Miscellaneous. This Agreement constitutes the entire understanding of the parties with respect to the subject matter of this Agreement and merges all prior communications, representations, and agreements. This Agreement may be modified only by a written agreement signed by the parties. If any provision of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be construed under the laws of the State of Washington, USA, excluding rules regarding conflicts of law. The application the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded. The parties agree that the Uniform Computer Transactions Act or any version thereof, adopted by any state, in any form ("UCITA"), shall not apply to this Agreement, and to the extent that UCITA may be applicable, the parties agree to opt out of the applicability of UCITA pursuant to the opt-out provision(s) contained therein.

Icetips Alta LLC
3430 East Highway 101, Ste. #28
Port Angeles WA 98362
EMail: support@icetips.com
http://www.icetips.com

Part



Chapter 2 - Version History

2 Version History

This chapter shows you information about what has changed or been updated in each version and build.

2.1 2018

This chapter lists all releases in 2018 along with changes to classes, templates and documentation.

Releases in 2018:
Version 2018.10.2460.323 19 - Sunday, October 14, 2018

Version 2018.10.2460.323 - Sunday, October 14, 2018

Updates, features:

February 22, 2016	Added IsFolderWritable 72 to the Core Class 52.
May 14, 2016	Embed tree structure synchronized. All Icetips Utilities embeds now show up under "Icetips | Utilities"
May 14, 2016	Added a new extension template that can duplicate a window 463 so you can determine at runtime if you want to open a SDI or MDI version of the same window. The duplicate is exactly like the original except the SDI/MDI are reversed, i.e. if the original window is MDI, the copy is a SDI and if the original is a SDI, the copy is a MDI. This comes in handy where a window needs to be opened with or without an active application frame for MDI windows.
June 15, 2016	Added ReleaseWindow 216 to Progress Class 205. This method can be called to make the window responsive during a tight loop operation.
September 1, 2016	Added LineCnt to StringClass 280. This property contains the number of lines in the Lines 284 queue property.
November 29, 2016	Added "Window.Init - After Resize" as target embed for "Call procedure from all procedures 435" template. Generated into ThisWindow.Init at priority 8260.
August 25, 2018	Changed "Add Header Sort to Queue 458" template to show the queue and list names in the extension list.
October 3, 2018	Templates updated to use CHM help.
October 7, 2018	Templates updated to support **Clarion 11**

Fixes:

April 14, 2016	GenEnvVariables 263 would cause an "Array out of bounds" GPF if it was called in debug mode. Fixed.
May 14, 2016	Link to Live Support on the Support tab in the Utilities templates was not working correctly. Fixed.
August 1, 2016	CheckOplocs 339 and SetOplocksOff 353 did not always work correctly. Fixed.
November 21, 2017	GetNextQuarter 103 would return the wrong dates if the source date was in the last quarter. Fixed.
October 12, 2018	Regression: "Include Thread Limiter" in Icetips Utility Classes Global 451 generated incorrect definition in multi-dll applications. Fixed.

Releases in 2016:
Version 1.2446 22 - January 25, 2016

Releases in 2015:
Version 1.2.2441 24 - November 2, 2015
Version 1.2.2433 24 - June 29, 2015
Version 1.2.2430 25 - February 24, 2015

Releases in 2014:
Version 1.2.2427 27 - July 6, 2014
Version 1.2.2423 27 - January 27, 2014

Releases in 2013:
Version 1.2.2415 29 - May 14, 2013

Releases in 2012:
Version 1.2.2408 31 - July 5, 2012
Version 1.2.2406 32 - May 8, 2012

Releases in 2011:
Version 1.1.2397 34 - May 4, 2011

Releases in 2010:
Version 1.1.2394 35 - December 21, 2010
Version 1.1.2392 35 - December 1, 2010
Version 1.1.2390 37 - September 7, 2010
Version 1.1.2387 37 - June 6, 2010
Version 1.1.2374 38 - March 15, 2010
Version 1.1.2367 38 - February 26, 2010
Version 1.1.2359 38 - January 27, 2010
Version 1.1.2356 38 - January 26, 2010

Releases in 2009:
Version 1.1.2351 40 - April 15, 2009

Releases in 2008
Version 1.1.2320 41 - November 11, 2008
Version 1.1.2319 41 - September 02, 2008
Version 1.1.2316 41 - August 27, 2008

2.2 2016

This chapter lists all releases in 2016 along with changes to classes, templates and documentation.

Releases in 2016:

Version 2018.10.2460.323 - Sunday, October 14, 2018

Updates, features:

February 22, 2016	Added IsFolderWritable 72 to the Core Class 52.
May 14, 2016	Embed tree structure synchronized. All Icetips Utilities embeds now show up under "Icetips | Utilities"
May 14, 2016	Added a new extension template that can duplicate a window 463 so you can determine at runtime if you want to open a SDI or MDI version of the same window. The duplicate is exactly like the original except the SDI/MDI are reversed, i.e. if the original window is MDI, the copy is a SDI and if the original is a SDI, the copy is a MDI. This comes in handy where a window needs to be opened with or without an active application frame for MDI windows.
June 15, 2016	Added ReleaseWindow 216 to Progress Class 205. This method can be called to make the window responsive during a tight loop operation.
September 1, 2016	Added LineCnt to StringClass 280. This property contains the number of lines in the Lines 284 queue property.
November 29, 2016	Added "Window.Init - After Resize" as target embed for "Call procedure from all procedures 435" template. Generated into ThisWindow.Init at priority 8260.
August 25, 2018	Changed "Add Header Sort to Queue 458" template to show the queue and list names in the extension list.

Fixes:

April 14, 2016	GenEnvVariables 263 would cause an "Array out of bounds" GPF if it was called in debug mode. Fixed.
May 14, 2016	Link to Live Support on the Support tab in the Utilities templates was not working correctly. Fixed.
August 1, 2016	CheckOplocs 339 and SetOplocksOff 353 did not always work correctly. Fixed.
November 21, 2017	GetNextQuarter 103 would return the wrong dates if the source date was in the last quarter. Fixed.

Version 1.2.2446 - January 25, 2016

Updates, features:

November 12, 2015	Added FormatXML 296 method to String Class and GetXMLDateTime 70 method to Core Class
November 20, 2015	Updated the Add Manifest 432 template to support windows 10.
January 15, 2016	Added EncodeXML 294 method to String Class.
January 15, 2016	Installer and everything else is now correctly dual-code signed, using both SHA1 and SHA2 code signing certificates.

Fixes:

November 20, 2015	Updated the Add Manifest 432 template to support windows 10.
January 20, 2016	Export template for Clarion template chain (Legacy) was not reliably

updated. It is now updated automatically with Build Automator every time a new build is created. Fixed.

Releases in 2015:
Version 1.2.2441 24 - November 2, 2015
Version 1.2.2433 24 - June 29, 2015
Version 1.2.2430 25 - February 24, 2015

Releases in 2014:
Version 1.2.2427 27 - July 6, 2014
Version 1.2.2423 27 - January 27, 2014

Releases in 2013:
Version 1.2.2415 29 - May 14, 2013

Releases in 2012:
Version 1.2.2408 31 - July 5, 2012
Version 1.2.2406 32 - May 8, 2012

Releases in 2011:
Version 1.1.2397 34 - May 4, 2011

Releases in 2010:
Version 1.1.2394 35 - December 21, 2010
Version 1.1.2392 35 - December 1, 2010
Version 1.1.2390 37 - September 7, 2010
Version 1.1.2387 37 - June 6, 2010
Version 1.1.2374 38 - March 15, 2010
Version 1.1.2367 38 - February 26, 2010
Version 1.1.2359 38 - January 27, 2010
Version 1.1.2356 38 - January 26, 2010

Releases in 2009:
Version 1.1.2351 40 - April 15, 2009

Releases in 2008
Version 1.1.2320 41 - November 11, 2008
Version 1.1.2319 41 - September 02, 2008
Version 1.1.2316 41 - August 27, 2008

2.3 2015

This chapter lists all releases in 2015 along with changes to classes, templates and documentation.

Releases in 2015:

Version 1.2.2441 - November 2, 2015

Updates, features:

July 5, 2015	Added MatchControlSize 74 method to core class.
September 12, 2015	Added pDel parameter to StringToLines 313, allowing passing a delimiter to split the string to lines by.
September 12, 2015	Added pQuoteChar parameter to ParseCSVLine 305, allowing setting the quote char for strings.
September 12, 2015	Added pDel parameter to FileToLines 294, allowing it to parse files with End-Of-Line delimiters other than CR, LF and CR+LF.
September 12, 2015	Added pDel parameter to WriteQToFile 316, allowing setting the End-Of-Line delimiters when writing the Lines queue to file. By default CR+LF (<13,10>) is used.
October 20, 2015	Added LinesToFile 302 method to StringClass 280. It's a wrapper for WriteQToFile.
October 20, 2025	Added StringToFile 312 method to StringClass 280. It's a wrapper for WriteStringToFile.
October 26, 2015	Added SkipEOLOnLastLine 285 property to StringClass. When using WriteQToFile or LinesToFile it will create an empty line because it added EOL at the end of the last line. If SkipEOLOnLastLine is set to True, then it does not add the EOL at the end of the last line. Original behaviour (adding an empty line) is left intact so it does not break code.
October 28, 2015	Added GetStringBetween 300 method to StringClass 280. It extracts a string from another string by searching for a beginning and end string, optionally starting at given positions. Can be specified if it's case sensitive or not.
October 29, 2015	Added SetStringBetween 309 method to StringClass 280. It inserts a string from another string in the same way as GetStringBetween 300 does.

Fixes:

July 4, 2015	Added ADJUST attribute to all #SHEET statements in the templates to make sure they adjust (more) properly when fonts in the IDE are changed.
August 14, 2015	StringToLines 313 would fail if the string contained no end-of-line characters. Fixed.
September 12, 2015	End-Of-Line variable in LinesToString 303 was declared as CSTRING(4) instead of CSTRING(5). Fixed.
October 20, 2015	Preserve Variables template 474 did not generate the correct code if conditions were used. Fixed.

Version 1.2.2433 - June 29, 2015

Updates, features:

June 22, 2015	Added pDelimterStartsLine parameter to SplitString 311. Allows it to split lines based on the start characters of a line, not end characters.
June 29, 2015	Added ParseCSVLine 305 method to parse a line with CSV data. Also added FreeCSVFields 297 (protected) method.

Fixes:

March 10, 2015	The template could cause classes to export an older version which could cause "xxx is unresolved for export" errors when compiling exporting dlls. Fixed.
May 24, 2015	Fixed a potential memory leak in the AddIntoParenthesis 287 method.
June 27, 2015	Image Class ResizeImage method did not work correctly on reports. Fixed.
June 27, 2015	Image Class ResizeImage did not correctly size images when it was called multiple times for the same image control. Fixed.

Version 1.2.2430 - February 24, 2015

Updates, features:

August 21, 2014	Added two new template files, ITUtilExt.tpw and ITUtilUtil.tpw. The first one will be used for additional Extension templates, the second for Utility templates.
December 3, 2014	Added FormatDate and FormatTime to DateClass. Those take a format string, such as 'mm-dd-yyyy' and translate the passed date into the format passed in. This makes the formatting of dates much more flexible than the picture formats in Clarion.
December 4, 2014	Added pCaseSensitive=0 to CountFinds 59, FindReplace 61 and SearchReplace 77. Previously those were case insensitve, but can now handle case sensitive searches.
December 16, 2014	Added Lesser 73, Greater 74 to the Core Class 52 and ColorToRGB 340 and GetColorBrightness to the Utilities Class 335.

Fixes:

July 16, 2014	RemoveHTML 308 did not remove white spaces at beginning or end of the resulting string. Fixed.
July 16, 2014	RemoveHTML 308 did not remove Fixed.
July 21, 2014	Preserve Variable Data 474 template could fail to generate an END statement into ThisWindow.Kill in some circumstances. Fixed.
October 1, 2014	Demo app for the new IDE (UtilDemoC7.app) was missing a file, UtilDemo. Version in the install. Fixed.
December 19, 2014	FileErrorCode() can return alphanumeric values since it returns a string. ErrorMsg 343 always interpreted it as a LONG so results could be incorrect. Fixed.

Releases in 2014:
Version 1.2.2427 27 - July 6, 2014
Version 1.2.2423 27 - January 27, 2014

Releases in 2013:
Version 1.2.2415 29 - May 14, 2013

Releases in 2012:
Version 1.2.2408 31 - July 5, 2012
Version 1.2.2406 32 - May 8, 2012

Releases in 2011:
Version 1.1.2397 34 - May 4, 2011

Releases in 2010:
Version 1.1.2394 35 - December 21, 2010
Version 1.1.2392 35 - December 1, 2010
Version 1.1.2390 37 - September 7, 2010

Version 1.1.2387 [37] - June 6, 2010
Version 1.1.2374 [38] - March 15, 2010
Version 1.1.2367 [38] - February 26, 2010
Version 1.1.2359 [38] - January 27, 2010
Version 1.1.2356 [38] - January 26, 2010

Releases in 2009:
Version 1.1.2351 [40] - April 15, 2009

Releases in 2008
Version 1.1.2320 [41] - November 11, 2008
Version 1.1.2319 [41] - September 02, 2008
Version 1.1.2316 [41] - August 27, 2008

2.4 2014

This chapter lists all releases in 2014 along with changes to classes, templates and documentation.

Releases in 2014:

Version 1.2.2427 - July 6, 2014

Updates, features:

March 29, 2014	Added EnumRegistryValues 227 method to the Registry Class 222.
April 8, 2014	Added checks to allow condition to apply on reading and/or writing in the Preserve Variable Data 474 template.
April 8, 2014	Added embeds before and after reading and writing each variable in the Preserve Variable Data 474 template.
April 8, 2014	Updated and improved visual information on the Preserve Variable Data 474 template.
May 30, 2014	Added support for CPCS reports to the Preserve Variable Data 474 template.
June 26, 2014	Added InsertString 301 to String Class.
June 27, 2014	Added Alias Files 448 global extension template. It adds PROP:Alias to all files in procedures.

Fixes:

January 30, 2014	Installer did not allow selecting the Clarion root install folder. Fixed.
February 21, 2014	The Icetips Preserve Variable Data 474 template did not work in source procedures. Fixed. Generated code is at priority 1 and priority 9999.
February 24, 2014	The pAppend parameter on WriteStringToFile 316 was not implemented. Fixed.
March 5, 2014	Example code for StringToWords 314 did not show access to the Word queue. Fixed
March 30, 2014	CompileSBProject 243 method uses Exists() without ShortPath(). This can cause problems in certain situations reporting that a file/folder does not exist when it does. Fixed.
June 16, 2014	Int64 variable declared in methods in the Registry Class 222 caused "Duplicate label" error. Fixed.

Version 1.2.2423 - January 27, 2014

Updates, features:

September 23, 2013	Added two parameters to CopyFiles 261 to allow further refining of the SHFileOperation attributes.
January 15, 2014	Added GenerateFileQueue 443 extension template.
January 27, 2014	Updated all template descriptions so they are prefixed with "Icetips Utilities: " to make it easier to find them!
January 27, 2014	Installer updated to work with Clarion 9.1

Fixes:

August 5, 2013	Icetips SQL Queue Process template did not use external names for the queue fields. Fixed.
October 25, 2013	Modified the "Preserver Variable Values" template code. The save was put in the wrong place in ThisWindow.Kill. Fixed.
December 2, 2013	SplitString 311 would fail for the last item in the string if it was just one character. Example "123;234;345" would work fine, where as "123;234;3" would correctly parse the first two elements, but the last one would be empty. Fixed.
December 12, 2013	Increased number of possible enumerated processes in EnumProcesses from 1024 to 10240.

Releases in 2014:
Version 1.2.2427 27 - July 6, 2014
Version 1.2.2423 27 - January 27, 2014

Releases in 2013:
Version 1.2.2415 29 - May 14, 2013

Releases in 2012:
Version 1.2.2408 31 - July 5, 2012
Version 1.2.2406 32 - May 8, 2012

Releases in 2011:
Version 1.1.2397 34 - May 4, 2011

Releases in 2010:
Version 1.1.2394 35 - December 21, 2010
Version 1.1.2392 35 - December 1, 2010
Version 1.1.2390 37 - September 7, 2010
Version 1.1.2387 37 - June 6, 2010
Version 1.1.2374 38 - March 15, 2010
Version 1.1.2367 38 - February 26, 2010
Version 1.1.2359 38 - January 27, 2010
Version 1.1.2356 38 - January 26, 2010

Releases in 2009:
Version 1.1.2351 40 - April 15, 2009

Releases in 2008
Version 1.1.2320 41 - November 11, 2008
Version 1.1.2319 41 - September 02, 2008
Version 1.1.2316 41 - August 27, 2008

2.5 2013

This chapter lists all releases in 2013 along with changes to classes, templates and documentation.

Releases in 2013:

Version 1.2.2415 - May 14, 2013

Updates, features:

September 20, 2012	Added GetBit 62, SetBit 78 and GetBitString 155 to the Core Class 52. This comes in very handy when using bitmap values!
September 21, 2012	Added ReadFileToQ 306 method to the String Class 280. It just calls FileToLines 294, but better matches naming convention with WriteQToFile 316.
September 21, 2012	Added GetLastInputTime 387 and GetIdleTime 386 to the Windows Class 365. Documentation has also been added.
September 22, 2012	Added option to Window Wizard 479 to create tabs on the window. You can enter the text for each tab and create as many tabs as you want.
April 15, 2013	Added "Icetips Preserve Variable Data 474" procedure extension template.
May 14, 2013	Installer is now compatible with Clarion 9.

Fixes:

September 21, 2012	#508: Verified bug in Clarion 8 that prevents .TXD files from being exported using **#EXPORT(%DictionaryFile)** which is used in the Icetips Export App and Dct 442 global extension template. PTSS #39582: http://problemtracker.softvelocity.com/clarion8/UpdtCustBugsview.php?ID1=39582
September 21, 2012	#332: Export Class no longer requires the ASCII driver.
September 21, 2012	#332: Record Class no longer requires the ASCII driver. There is no longer any dependency on the ASCII file driver in the Utilities as it has been replaced with calls to methods in the String Class 280.
September 21, 2012	#523: Page of Pages template 426 erroneously called %ITPisCpcsReportTemplate group instead of %ITUisCpcsReportTemplate. Fixed.
September 22, 2012	Help was not implemented in the Window Wizard 479. Fixed.
October 19, 2012	Under some circumstances FileString 284 was not disposed in the String class causing memory leak. Fixed
January 19, 2013	Zendesk support ticket #15: StringToWords 314 ignored words made up of digits only. Added parameter to StringToWords 314 to indicate if digits should count as words. Fixed.
February 25, 2013	Added support for Setup Builder 8 to the SetupBuilderClass 234.

Releases in 2012:

Releases in 2011:

Releases in 2010:

Version 1.1.2374 - March 15, 2010
Version 1.1.2367 - February 26, 2010
Version 1.1.2359 - January 27, 2010
Version 1.1.2356 - January 26, 2010

Releases in 2009:
Version 1.1.2351 - April 15, 2009

Releases in 2008
Version 1.1.2320 - November 11, 2008
Version 1.1.2319 - September 02, 2008
Version 1.1.2316 - August 27, 2008

2.6 2012

This chapter lists all releases in 2012 along with changes to classes, templates and documentation.

Releases in 2012:

Releases in 2011:

Releases in 2010:

Releases in 2009:

Releases in 2008

Version 1.2.2408 - July 5, 2012

Updates, features:

June 13, 2012	Added global template to handle Legacy applications. Use the "Icetips Utilities Classes Global - LEGACY ONLY" Global template instead of the "Icetips Utilities Classes Global - ABC ONLY" in the applications where you use the Icetips Utilities Classes. This template also needs to be added to the root/exporting dll app.
June 22, 2012	Added QueueResorted virtual method to the AddHeaderSortToQueue 458 template. This can only be implemented by modifying the ListQueue property in the brwext.inc file from being PRIVATE to PROTECTED. QueueResorted method is called after the queue is resorted, but a bug in the brwext.clw causes it to be called _before_ the queue is sorted so it's always one sort behind! The overridden method sorts the queue so after the parent call the queue is sorted correctly before your code executes.
June 22, 2012	Modified installer to change the ListQueue property from PRIVATE to PROTECTED.
June 22, 2012	Completed documentation for List procedures with Windows without HLP attribute 484 utility template.
June 22, 2012	Added TreatEmptyLastItemAsLine 285 property to the StringClass 280. This allows SplitString 311 to determine how to handle empty items at the end of the item array.
June 23, 2012	Added 5 new embed points to the Limit Program Instance 454 template to make it more flexible.
June 23, 2012	Completed documentation for Limit Program Instance 454 template.

June 23, 2012	Completed documentation for all Utilities templates 479.
June 25, 2012	Added a check to SetPageOfText 199 method in Page of Pages class 192 to prevent it from potentially replace the page-of token if it is found more than once in the metafile, indicating that the file contained the particular byte array in its internal code. This would prevent any problems due to replacing some of the meta file code from ever happening.
June 26, 2012	Finished documentation for Page of Pages Class 192.
June 27, 2012	Finished documentation of Write Version info to INI file template 456.
June 27, 2012	Finished documentation of Add Header Sort to Queue template 458.
June 27, 2012	Finished documentation of Bind/Unbind local variables template 460.
June 27, 2012	Finished documentation of Icetips Browse Checkbox update template 465.
June 28, 2012	Added and implemented the Icetips ShowFileRecord Wizard 488 utility template.
June 30, 2012	Finished documentation of Code templates 417.
July 2, 2012	Finished documentation of Page of Pages control template 426.
July 2, 2012	Finished documentation of all Utility templates 479.

Fixes:

June 9, 2012	ShowProgress 218 method in ProgressClass 205 would apply "vista fix" to progress step on all operating system versions. It could cause back jumping in the progressbar on XP under some circumstances. Now only applied if running under Vista or newer operating systems. Fixed.
June 22, 2012	SplitString 311 could fail with last item in the item array if it contained only one character. Fixed.
June 25, 2012	FindInString 295 method in the String Class 280 was not documented. Fixed.
June 26, 2012	Limit Program Instance template 454 didn't preserve variable setting for entries, i.e. if "!VarName" was used as "Unique Program String" it would be changed to "VarName". Same with the "Window caption string", "Message Caption" and "Message Text" entries. Fixed.
June 26, 2012	Some methods were missing from the Overview 280 and Methods 287 list in the String Class 280 chapter. Fixed.
June 27, 2012	GetThemedPanelFEQ 395 did not return the correct FEQ. Fixed.
June 29, 2012	ThemeAPanel 410 did not work in Clarion 8. Fixed.
June 29, 2012	ThemeAPanel 410 required XPTheme to be included for Clarion 8. Fixed.
June 29. 2012	Fixes to ThemeAPanel 410 for Clarion 8 did not work in Clarion 6. Fixed.

Version 1.2.2406 - May 8, 2012

Updates, features:

June 9, 2011	Added FileToLines 294 method to the String Class 280
September 20, 2011	Added GetRegEx 227 method to Registry Class 222
September 20, 2011	Added QueryValue 229 method to Registry Class 222
September 20, 2011	Modified OpenRegistryKey 229 method in Registry Class 222
September 20, 2011	Added GetValueType 228 method to Registry Class 222
September 27, 2011	Added IsProgramRunning 398 method to Windows Class 365
October 1, 2011	Added "Limit Program Instance 454" global extension template.
October 1, 2011	Added ActivateWindow 378 method to Windows Class 365
October 22, 2011	Added Export Windows without Help ID 484 utility template.
November 12, 2011	**Modified return value from the ITRun 269 method** so that if the called process returns no exit code, the method returns 0 instead of the return value from WaitForSingleObject. The reason for this change is that it is possible that if the called process returns nothing and WaitForSingleObject returns a value that the calling code is expecting, the results are wrong.
January 14, 2012	Finished documentation for Progress Class 205.

May 3, 2012	Added FindReplace 61, AllocateSearchString 58 and CountFinds 59 methods to Core class 52. FindReplace 61 is an adidtional search and replace method that returns the replaced string unlike SearchReplace 77 method which needs the string to be passed in as a variable.

Fixes:

October 6, 2011	GetMonthName 100 returned space padded month name. Fixed.
November 6, 2011	GetSBExecutable 248 in Setup Builder Class 234 would return true if it didn't find the exe. Fixed.
November 11, 2011	GetValueType 228 in Registry Class 222 did not return the value data type for the registry entry. Fixed.
November 11, 2011	GetRegEx 227 in Registry Class 222 did not handle value data type correctly. Fixed.
January 23, 2012	EnumRegistrySubKeys 226 did not implement the pIs64Bit parameter to enforce default, 32 or 64 bit keys. Fixed
April 25, 2012	Thread limiter Global 330 class instance was not exported correctly in multi-dll applications. Fixed.
May 4, 2012	The ITUtility.hlp file was not included in the install. Fixed.
May 4, 2012	The ITUtility.hlp was not activated in the main template file - ITUtility.tpl - so it did not open when using the "Help" button in the Clarion IDE! Fixed.

2.7 2011

This chapter lists all releases in 2011 along with changes to classes, templates and documentation.

Releases in 2011:

Releases in 2010:

Releases in 2009:

Releases in 2008

Version 1.1.2397 - May 4, 2011

Updates, features:

May 4, 2011	Install built for Clarion 8.0.

Fixes:

January 5, 2011	SplitString 311 could cause "Index Out of Range" in some cases if the last line was empty. Fixed.
January 12, 2011	FileToString 295 method in String Class 280 did not have the PROC attribute. Fixed.
February 23, 2011	ErrorMsg 343 did not return FileError() but Error() if it was called with (False, True) i.e. when it should only return FileErrorCode() and FileError(). Fixed.

2.8 2010

This chapter lists all releases in 2010 along with changes to classes, templates and documentation.

Releases in 2010:

Releases in 2009:

Releases in 2008

Version 1.1.2394 - December 21, 2010

Updates, features:

Fixes:

December 10, 2010 — EnumRegistrySubKeys would fail under windows 2008 server.

Version 1.1.2392 - December 1, 2010

Updates, features:

September 7, 2010 — Added ClassName 367 field to the ChildWindowQ 367 queue structure, used by the Windows Class 365.

September 9, 2010 — Added ITWinVirtualKeys.inc file which includes equates for all virtual keys as defined by winuser.h. This file is being included in the ITWin32Equates. inc file.

September 9, 2010 — Added equates for keyboard, mouse and hardware processing. This is being used in the upcoming Keyboard class.

Key events:
IT_KEYEVENTF_KEYDOWN
IT_KEYEVENTF_EXTENDEDKEY
IT_KEYEVENTF_KEYUP

Input options:
IT_INPUT_MOUSE
IT_INPUT_KEYBOARD
IT_INPUT_HARDWARE

IT_INPUT_CHAR
IT_INPUT_STRING

September 10, 2010	ID field to the ChildWindowQ 367 queue structure, used by the Windows Class 365. This is being retrieved in the EnumChildWindowProc 413 and EnumTopWindowProc 412 with IT_GetWindowLong(hwnd,IT_GWL_ID). This makes it possible to 100% identify a control and the ID is persistent.
November 21, 2010	Documentation done for File Select Class 128.
November 30, 2010	Documentation done for Files Class 167.
November 30, 2010	Added keywords to the Network Class 179.
November 30, 2010	Added ConvertToUNC 184 and GetUNCFileNames 349 to Network Class 179. Those methods simply call GetLocalNetworkFileName 186, but are easier to remember!
November 30, 2010	Added documentation for IT_NETRESOURCEs 181 and IT_NetworkShares 181 data types. HideDebugView 45 property and ConvertToUNC 184, EnumNetworkPrinters 186, GetLocalNetworkFileName 186, GetUNCFileName 188, IsLocalShare 189, IsUNC 190 and ShowLocalShares 191 methods in the Network Class 179.
November 30, 2010	Added documentation for "Icetips Hide_Windows while loading 450" global template.
December 1, 2010	Added documentation for "Add Vista/Win7 Manifest to application 432" global template.
December 1, 2010	Updated screenshots, keywords and information for several global extension templates.

Fixes:

September 7, 2010	ProcessID 367 property of the ChildWindows 368 queue was not being primed in the EnumChildWindowsProc in Windows Class 365. Fixed.
November 24, 2010	Install did not place uninstall log in the appropriate location. Fixed. Old uninstall files are moved to the proper location before the new install log is started.
November 29, 2010	The GetLastWeek 96 and GetThisWeek 113 did not return correct dates in some cases. Fixed.
November 29, 2010	The GetLastWorkWeek 97 and GetThisWorkWeek 114 did not return the correct dates in some cases. Fixed.
November 29, 2010	The GetLastWorkWeek 97, GetThisWorkWeek 114 and GetNextWorkWeek 106 did not return the correct end date, i.e. it would return Saturday rather than Friday. Fixed.
December 1, 2010	The TREE embed structure in Thread limit template 471 caused it to create a second "Local Object" instance in the embed tree. Fixed.

Version 1.1.2390 - September 7, 2010

Updates, features:

July 2, 2010	Documentation done for File Search Class 134.
July 24, 2010	22 new methods added to the Date Class: GetThisWeek 113, GetLastWeek 96, GetNextWeek 104, GetThisWorkWeek 114, GetLastWorkWeek 97, GetNextWorkWeek 106, GetThisMonth 111, GetLastMonth 94, GetNextMonth 102, GetThisQuarter 112, GetLastQuarter 95, GetNextQuarter 103, GetThisYear 115, GetLastYear 98, GetNextYear 107, GetLast12Months 92, GetMonthToDate 101, GetQuarterToDate 110, GetYearToDate 119, GetMonthFromDate 99, GetQuarterFromDate 109, GetYearFromDate 118. Documentation for Date Class 86 updated.
July 24, 2010	Documentation done for Date Class 86.
July 28, 2010	6 new methods added to the Date Class: DateDiff 90, DateAdd 90 and GetDate 91 to get date differences and SetWeekStartDay 122, GetWeekStartDay 117 and GetWeekFirstDay 116 that deal with different start day of a week. By default the week start day is set to Monday, but can now be set to any day. Documentation for Date Class 86 also updated.
September 3, 2010	Added FreeFiles parameter to the ScanFiles 143 method in the File Search Class 134. This allows the method to be called repeatedly for different folders or different wildcards without having to handle the files in the Files queue until all the files have been read.
September 3, 2010	Added attributes parameter to the FileExists 60 method to Core Class 52. This allows the method to be used to find more than just FF_:Normal files, such as directories, hidden files etc.
September 5, 2010	Added FindInString 295 method to StringClass 280.
September 6, 2010	Documentation done for INI Class 170.

Fixes:

July 21, 2010	HtmlToColor did only handle html color value strings that started with '#' Fixed so now it can handle html value colors without the '#' prefix. I.e. 'C1D8F0' instead of '#C1D8F0'
September 3, 2010	ScanFiles did not check if the path passed in ended with a backslash which caused an error. Fixed.
September 5, 2010	PROC attribute missing from SetFileAttrib 78 method in Core Class 52. Fixed.

Version 1.1.2387 - June 6, 2010

Fixes:

June 6, 2010: Installer updated for Clarion 7.2.
June 1, 2010: Problem with GPF in ITRun32C7.dll. Moved the dll from Borland C++ to Visual C++ which fixed the problem.
March 17, 2010: ITRun32.dll and ITRun32C7.dll were not compiled correctly. Fixed.
March 22, 2010: Information about FNS_Parts 53 and IT_GUID 53 in the Core Class 52 were switched. Fixed.
May 31, 2010: Date class would not always return the correct date name. Fixed.

Version 1.1.2374 - March 15, 2010

Updates, features:

March 11, 2010: Added IsUserAdmin 268 and IsProgramElevated 269 methods to the ShellClass 257.
March 4, 2010: Documentation done for Armadillo Class 45 and Armadillo Code Generator Class 49.
March 3, 2010: Added "Icetips: Prepare multi-DLL app" template. It lets you export global data to txd file ready to import into a dictionary. It also allows you to export procedures that are set to export as external procedures in a TXA file. This way you can export them as external files in TXA and then import them into apps that need to use those procedures as external procedures.

Fixes:

March 3, 2010: Install did not correctly replace PRIVATE property on INIFile critical section in ABUTIL.INC. Fixed
March 3, 2010: Install did not handle ITRun32.dll and ITRun32C7.dll install correctly. Fixed.

Version 1.1.2367 - February 26, 2010

Updates, features:

February 17, 2010: Added GetDriveSerialNumber 131 and GetVolumeInfo 132 methods to File Class 128
February 17, 2010: Documentation for File Class 128 completed.
February 20, 2010: Thread Limiter class and template in Alpha testing.

Fixes:

February 11, 2010: ProgressClass would not show a full progress on Vista. This was a known problem with Vista and later machine where MS changed the way progress bars were updated on themed windows. The trick is to update the progressbar then take it backward one notch. Fixed.

Version 1.1.2359 - January 27, 2010

Updates, features:

February 2, 2010: Added CheckOplocks and SetOplocksOff methods to the Utility 335 Class.

Version 1.1.2356 - January 26, 2010

Updates, features:

1. String class 280 would set LastAPIError 55 and LastAPIErrorCode 56 to the error returned by WriteFile rather than CreateFile if CreateFile failed. Fixed.
2. String class 280 would set LastAPIError 55 and LastAPIErrorCode 56 to the error returned by ReadFile rather than CreateFile if CreateFile failed. Fixed.

3. Added DumpLinesInQ 293 method to String class 280 to make it easy to move contents of the Lines queue property to a local queue
4. LongToHex 73 moved from UtilityClass 335 to CoreClass 52
5. Added URLEncode method.
6. Added parameter to SelectFile method in the FileSelectClass. This allows to passing the entire filename to the function and suggesting the passed filename rather than just the folder name.
7. Added Clarion Resources, including what is added by the "Add Compile Date/Time to version 430" template, to the Version class. 355
8. Added CreateFolder 262 method to the ShellClass 257. It simply wraps the CreateDirectory 262 in a new name.
9. Added IsAppframe 70 method to the CoreClass 52. It returns true if the current target is an appframe window.
10. ITINIClass 170 documentation finished.
11. GetFileSize 66 method added to the CoreClass 52. The method supports > 2GB file sizes.
12. Added ITFileQueueLS queue type which uses a DECIMAL(15,0) for the file size. This structure is NOT compatible with DIRECTORY, but can be used to store similar information with large file sizes. See GetFileSize 66.
13. Added pLocation parameter to GetLastAPIError in CoreClass 52. This allows passing a location string that is shown in the output text.
14. Added GetWeekNumber 117 method to DateClass 86 that uses ISO8601 to calculate the current week number.
15. Added FileExists 60 method to CoreClass 52.
16. Added ByteToHex 59 method to CoreClass 52.
17. Added AppendToLine 289 method to String Class 280.
18. Added SetLineValue 308 method to String Class 280.
19. Added GetWord 301 method to String Class 280.
20. Added SetPunctuationString 310 method to String Class 280.
21. Added SetSplitStringProgress 310 method to String Class 280.

Fixes:

1. ODS 75 method used a fixed 2K string. This could cause confusion if the output string was bigger and only the first 2K showed up in Debug View. Modified to contain all the data passed into the method. Fixed.
2. AssociateProgram 260 method in Shell class was not working correctly. Fixed.
3. GetAssociatedProg 264 could fail if extension was more than 4 characters long including the dot prefix. Fixed.
4. AddLine 288 method had a potential memory abuse problem that would cause a GPF and out of memory errors when building very large queues. Fixed.
5. SearchReplace 77 could fail to replace if the replace string was shorter than the search for string. Fixed.
6. GetSpecialFolder 267 used 0{PROP:Handle} to get a window handle. However since this method can be called before any window has been opened it could cause a GPF. Changed to use the desktop window instead. Fixed.
7. Standardized Window Code 514 template didn't generate code correctly for the data and routine embeds (an extra couple of spaces was added to the start of the TXA code). Fixed.
8. ReadDirectories 142 method in the File Search Class 134 counted "." and ".." files. Fixed.
9. File Search Class 134 used LONGs for files sizes and totalled file sizes. Changed to Decimal(15,0).
10. ScanFiles method in the File Search Class 134 did not return the correct number of files. Fixed.
11. PathIsDir 273 in CoreClass did not call the correct API. Fixed.
12. Added PROC attribute to GetLastAPIErrorCode 67 method.
13. Added ReturnEmpty parameter to ExpandEnvString 263 method in Shell Class 257.

2.9 2009

This chapter lists all releases in 2009 along with changes to classes, templates and documentation.

Version 1.1.2351 - April 15, 2009

Updates, features:
1. Utility class 335 documentation finished.
2. String class 280 documentation finished.
3. All topics in the help show when they were last updated. Does not show in the PDF though.
4. RemoveForwardSlash 77 and RemoveBackSlash 76 did not work correctly if the string started and ended with the slash and both should be removed. Fixed.
5. GetFilePart 65 and RemoveBackSlash 76 added to Core Class Demo 513.
6. Added "pParameters" parameter to ITRunFile 270 to allow passing of parameters to it.
7. Added SetEnvVar 274 method to Shell Class 257 to allow setting environment variables - needs testing.
8. Added ExpandEnvString 263 method to Shell Class 257 to allow expanding environment variables.
9. Added demo of ExpandEnvString 263 to both expand a variable in a string and also us ITRun with %SystemRoot%\System32\taskmgr.exe which runs the task manager.
10. Added code template 418 to call the GetSpecialFolder 267 method of the Shell Class 257.

Fixes:
1. In some circumstances the SplitString 311 method of the string class 280 could cause a GPF because the end of the datastream was miscalculated as negative. Fixed.
2. ReadFileToString 307 method in the string class was opening the file to read in read+write mode. Changed to read mode only.
3. The OL 284 member of the Lines queue was declared as Cstring(1025) and thus restricted to 1K of data. It is now dynamic.

   NOTE: IF you have worked with the Lines queue manually where you have used **Free(ITS.Lines)** you now need to call the **FreeLines** 298 method instead of Free(ITS.Lines) or the equivalent **or you risk memory leaks.** Fixed.

2.10 2008

This chapter lists all releases in 2008 along with changes to classes, templates and documentation.

Version 1.1.2320 - November 11, 2008

Updates, features:

1. Utility class 335 documentation finished.
2. String class 280 documentation finished.
3. All topics in the help show when they were last updated. Does not show in the PDF though.
4. RemoveForwardSlash 77 and RemoveBackSlash 76 did not work correctly if the string started and ended with the slash and both should be removed. Fixed.
5. GetFilePart 65 and RemoveBackSlash 76 added to Core Class Demo 513.

Version 1.1.2319 - Tuesday, September 02, 2008

Updates, features:

1. New method in CoreClass, RemoveForwardSlash

Fixes:

1. MS Header template had the wrong image size. Fixed.
2. tThemedControls was missing the TYPE attribute. Fixed.
3. "Add Header Sort to Queue" template made use of a class property (UsePictureForCase) that was not in Clarion 6.1 causing compile errors in Clarion versions prior to Clarion 6.2. Fixed.
4. SelectFile in the FileSelectClass 148 could clear variable when the FileDialog was canceled. Regression in build 2316. Fixed.
5. ITUtilityClass.inc file was not update with current version number. Fixed.
6. SearchReplace 77 method in the CoreClass 52 would fail when searching for '/' when the string to be searched contained '//' - only the first character would be replaced. Fixed.
7. SearchReplace 77 method in the CoreClass 52 would fail when searching for '/' and replacing with '//' - it would fill the entire string with '/' from the first occurance. This only happens if the replace string was longer and if both search and replace contained all the same character. Fixed.

Version 1.1.2316 - August 27, 2008

Updates, features:

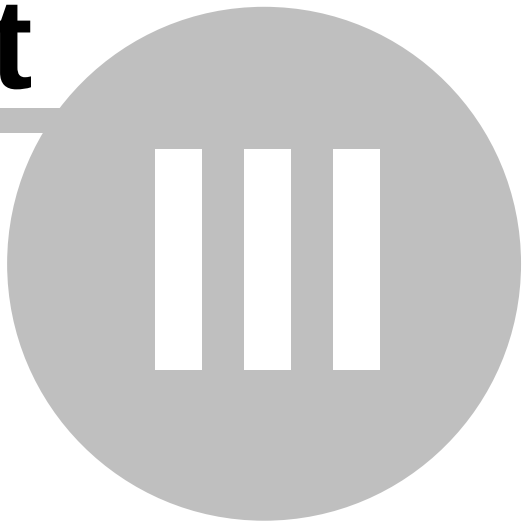
2. Core Class documentation finished.
3. Demo apps will now be done for each individual class, rather than for the whole project as it get's way too big.
4. Core Class demo application finished.
5. LastApiError 55 and LastApiErrorCode 56 properties added to class. They are set in the GetLastAPIError 66 and GetLastAPIErrorCode 67
6. Install improved and tested on a clean Clarion 6.3 9059 platform.
7. Build Automator script created to automatically handle new builds.

Fixes:

8. GetFileAttrib 64(String...) was not passing the string on to the GetFileAttrib 64(*CString...) method, causing it to fail since no filename was passed to it. Fixed.

9. GetLastApiError 66 and GetLastAPIErrorCode 67 methods moved from WindowsClass 365 to CoreClass 52
10. ITWinWiz.tpl contained a reference to a template that was not included with the Utilities. Fixed.

Part



Chapter 3 - Classes

3 Classes

There are 30 classes in the Icetips Utility Classes right now. These are classes that we have been using in our own development work starting around 2003. The Icetips Utility Classes are standard ABC classes. In Beta 3 a Global extension template was added to implement the classes in the application. As documentation and development continues we will add class templates so the classes can be easily added to procedures as well as to create derived classes and enable overriding methods. Some of the classes have not been fully documented yet.

3.1 Armadillo Class

3.1.1 Overview — Armadillo Class

This class is a wrapper for some of the functions in the ArmAccess.dll from Silicon Realms. This class requires that you add _ITARM_ => 1 to your project defines and also add the IT_ARMACCESS.LIB to your project's "Library, Object and resource file" section. If this is not done the methods in this class will throw an error message when you attempt to call them.

```
ITArmadilloClass        Class(ITShellClass 257
),TYPE,Module('ITArmadilloClass.clw'),Link('ITArmadilloClass',_ITUtilLinkMode_),DLL
(_ITUtilDllMode_)
HideDebugView 45            Byte

InstallKey 46               Procedure(STRING pName, STRING pCode), BYTE ! Returns
true/false if the key was valid
NotCompiledMessage 46       Procedure(String pS)
PTD 47                      Procedure(String pS, Byte pHideDebug=False),VIRTUAL
ShowEnterKeyDialog 47       Procedure(), BYTE ! Returns true/false if a key was
entered
UpdateEnvironmentVars 47    Procedure
Construct 48                Procedure
Destruct 48                 Procedure
                        End
```

3.1.2 Properties — Armadillo Class

The Armadillo Class has only one property, HideDebugView 45.

```
HideDebugView 45            Byte
```

3.1.2.1 HideDebugView — Armadillo Class - Properties

The HideDebugView is set to either True or False in the Construct 48 method.

It is useful during debugging, but for safety reasons it is actually commented completely out in the class to prevent the application accidentally to send information about keys to tools such as DebugView!

To enable debug output, you must change:

Construct:

Comment out the last line:

SELF.HideDebugView = True !! Comment out

Other methods:

Search for "!SELF.PTD" and replace with "SELF.PTD"

I.e. uncomment the SELF.PTD.

Make sure that before you deploy your program, that you turned the debug output OFF!

3.1.3 Methods Armadillo Class

There are currently 7 methods in the Armadillo Class.

3.1.3.1 InstallKey Armadillo Class - Methods

Prototype: **(STRING pName, STRING pCode), BYTE !**

pName The owner of the license key
pCode The license key

Returns Returns true/false if the key was valid

This method calls the InstallKey function in the ArmAccess.dll to verify the license key.

Example:

```
ITA  ITArmadilloClass
 Code
 If ITA.InstallKey(Loc:Name,Loc:Key)
   ! Ok, it is valid now!
 Else
   ! Code is invalid.
 End
```

See also:
ShowEnterKeyDialog 47
UpdateEnvironmentVars 47

3.1.3.2 NotCompiledMessage Armadillo Class - Methods

Prototype: **(String pS)**

pS Method name calling NotCompiledMessage

This method is called by any of the other methods if the _ITARM_ define is not correctly set in your application. This method is only called by other methods in the class.

3.1.3.3 PTD

Armadillo Class - Methods

Prototype: **(String pS, Byte pHideDebug=False),VIRTUAL**

pS String to send to OutputDebugString

pHideDebug Indicates if the debug output should not be sent (hidden) to OutputDebugString .

This method sends the string in pS to OutputDebugString.

Example:

```
ITA  ITArmadilloClass
 Code
 ITA.PTD('Debug information')
```

See also:
PTD 76 (Core)

3.1.3.4 ShowEnterKeyDialog

Armadillo Class - Methods

Prototype: **(), BYTE**

Returns Returns true/false if a key was entered

This method calls the ShowEnterKeyDialog function in the ArmAccess.dll and returns the value returned by that function. The window that is shown allows the user to enter the keycode. You do not need to call this method, you can simply create your own window in Clarion and let the user enter the license information there.

Example:

```
ITA  ITArmadilloClass
 Code
 If ITA.ShowEnterKeyDialog()
   ! A key was entered
 Else
   ! User cancelled.
 End
```

See also:
InstallKey 46

3.1.3.5 UpdateEnvironmentVars

Armadillo Class - Methods

Prototype: **(none)**

This method calls the UpdateEnvironment function in the ArmAccess.dll, which updates the Armadillo environment variables. This can be useful if you want to make sure that variables are updated before you check the EXPIRED variable to see if the program has expired. Since Armadillo cannot detect

automatically if the program expires as it is running, you need to check for it if the program might be run for days at the time without being closed down.

Example:

```
ITA  ITArmadilloClass
US   String(255)
EX   String(255)
 Code
 ITA.UpdateEnvironmentVars
 EX = ITA.GetEnvVar(265)('EXPIRED')
 US = ITA.GetEnvVar(265)('ALTUSERNAME')
 ITA.ODS(75)('  Expired  = ' & EX)
 ITA.ODS(75)('  Alt user = ' & US)
 If Clip(Upper(US)) = 'DEFAULT'
   !! This is a demo
 Else
   !! This is NOT a demo
 End
 If Clip(Upper(EX)) = 'TRUE'
   !! Expired
 Else
   !! Not expired
 End
```

See also:
InstallKey 46

ShowEnterKeyDialog 47

3.1.3.6 Construct

Prototype: **(none)**

The constructor checks to see if the commandline contains a /ITARMADILLO flag and if it does it sets HideDebugView 45 to False so that debug statement can be seen.

See also:
HideDebugView 45

3.1.3.7 Destruct

Prototype: **(none)**

There is currently no code in the destructor.

See also:
Construct 48

3.2 Armadillo Code Generator Class

3.2.1 Overview — Armadillo Code Generator Class

This class is ONLY to be used in programs that generate Armadillo keycodes. NEVER include this class in an application that includes Armadillo protection. We have tried to make it impossible to do, but please be warned. This class will only work if you add a _ITARMCODEGEN_=>1 to your project. It will not work if you have the _ITARM_ => 1 define in your project, which is required for the Armadillo Class 45.

This class only has one method in it, to generate a Short3Key keycode.

```
ITArmCodGenClass          Class(ITShellClass 257
),TYPE,Module('ITArmCodGenClass.clw'),Link('ITArmCodGenClass',_ITUtilLinkMode_),DLL
(_ITUtilDllMode_)
Template 49                    CString(255)
ExpireInDays 49                Long
CreateCodeShort3Key 50         Procedure(Byte pLevel, String pName, String pTemplate,
Long pDays=0),STRING
                          End
```

3.2.2 Properties — Armadillo Code Generator Class

The Armadillo Code Generator Class has only 3 properties which are more for external reference than internal use since there is only one method in the class.

```
Template 49                    CString(255)
ExpireInDays 49                Long
```

3.2.2.1 Template — Armadillo Code Generator Class - Properties

This is the template used for the code generation. This is the text that you use in the "Encryption Template" in the Armadillo program to encrypt your application with.

3.2.2.2 ExpireInDays — Armadillo Code Generator Class - Properties

This is the number of days until the certificate expires. If it is a permanent certificate, the ExpireInDays should be 0 (zero)

3.2.3 Methods — Armadillo Code Generator Class

The Armadillo Code Generator Class only contains one method.

```
CreateCodeShort3Key 50         Procedure(Byte pLevel, String pName, String pTemplate,
Long pDays=0),STRING
```

3.2.3.1 CreateCodeShort3Key

Prototype:	**(Byte pLevel, String pName, String pTemplate, Long pDays=0),STRING**
pLevel	The signature level for the key. Valid values are 1-10 both included.
pName	The user name to create the key for.
pTemplate	The encryption template that will be used to encrypt the key.
pDays	The number of days that the certificate is valid for. 0 (zero) indicates a permanent certificate
Returns	The key generated.

This method calls the CreateCodeShort3 function in the CodeGen.dll to generate the key. Note that this method does not support generating keys for hardware locked certificates. This method also sets the Template 49 and ExpireInDays 49 properties so you can query them after calling the method if you need to.

Example:

```
ITA ITArmCodGenClass
 Code
 Level      = 10                      !! Fixed to level 10
 Name       = CUS:Name                !! Customer name
 Template   = ARM:ProductEncTemplate  !! Product encryption template
 ProductKey = ITA.CreateCodeShort3Key (:Level, Name, Template)
 Display(?ProductKey)
```

3.3 Controls Class

3.3.1 Overview Controls Class

```
ITControlsClass
CLASS(ITStringClass),TYPE,Module('ITControlsClass.clw'),Link('ITControlsClass',_ITU
tilLinkMode_),DLL(_ITUtilDllMode_)
Controls                    &ITCtrlQ
RegisterWindow              Procedure(Byte pIncFrame=0)
GetControlText              Procedure(Long pFEQ),String
GetTypeText                 Procedure(Long pType),String
IsTranslatable              Procedure(Long pFEQ),Byte
GetControlByLabel           Procedure(String pLabel)
RemoveControlByLabel        Procedure(String pLabel,Byte pRemove=0)
RemoveControlList           Procedure(String pControlList,Byte pRemove=0)
CheckListbox                Procedure(Long pFEQ),Byte
Construct                   Procedure
Destruct                    Procedure
                          END
```

3.3.2 Methods Controls Class

Enter topic text here.

3.3.3 Properties Controls Class

Enter topic text here.

3.4 Core Class

3.4.1 Overview Core Class

The core class includes methods that can be used by other classes, basic methods that perform low level functions. It contains basic search method, file splitting functions, path functions and such. Any absolute core functions that we figure we may need will be added to the core class.

Please check out the short tutorial 83 at the end of this chapter.

```
ITCoreClass
Class,TYPE,Module('ITCoreClass.clw'),Link('ITCoreClass',_ITUtilLinkMode_),DLL(_ITUt
ilDllMode_)

DebugLevel 55                Byte
EXEName 55                   CString(1025)
FileParts 55                 Group(FNS_Parts)
                           End
ProgPath 56                  CString(10241)
ProgramCommandLine 56        CString(10241)
ProgramDebugOn 56            Byte
ComputerName 54              CString(IT_MAX_COMPUTERNAME_LENGTH+1)
UserName 54                  CString(256)
XPThemesPresent 57           Byte
LastAPIError 55              String(255)
LastAPIErrorCode 56          Long
ReplaceString 57             &String,PRIVATE
UrlStr 57                    &CString


AllocateSearchString 58      Procedure(Long pSize)
AllocateURLString 58         Procedure(Long pSize)
ByteToHex 59                 Procedure(Byte pByte),String
CountFinds 59                Procedure(String pFind, String pReplace, String
pSearchS),Long
CreateGUID 60                Procedure(Byte pAddBraces=1),STRING
FindReplace 61               Procedure(String pFind, String pReplace, String
pSearchS, <*LONG pNewLen>),String
FixPath 62                   Procedure(*CString pPath),String
FixPath 62                   Procedure(String pPath),String
GetComputerName 63           Procedure(),String,PROC         ! Returns the name of
the computer.  Puts it into the ComputerName property
GetFileAttrib 64             Procedure(*CString pFile, <*Byte pReadOnly>, <*Byte
pHidden>, <*Byte pSystem>),Long,PROC
GetFileAttrib 64             Procedure(String pFile, <*Byte pReadOnly>, <*Byte
pHidden>, <*Byte pSystem>),Long,PROC
GetFilePart 65               Procedure(String pFilename,Byte pPart),String
GetLastAPIError 66           Procedure(<*Long pErrorCode>),String
GetLastAPIErrorCode 67       Procedure(),Long
GetReplaceString 67          PROCEDURE(),STRING
GetTempFilename 68           Procedure(<String pPath>,<String pPrefix>),String
GetTempFolder 69             Procedure(),String
GetFilePart 65               Procedure(String pFilename,Byte pPart),String
GetUserName 69               Procedure(),String
GetXMLDateTime 70            PROCEDURE (ANY pDate, ANY pTime, <STRING
pTimeZone>),STRING
IsFileInUse 71               Procedure(*CString pFile),Byte
IsFileInUse 71               Procedure(String pFile),Byte
```

```
IsFolder 71                    Procedure(*CString pPath),Byte
IsFolder 71                    Procedure(String pPath),Byte
LongToHex 73                   Procedure(Long pLong),String
MatchControlSize 74            PROCEDURE (LONG pSourceControl, LONG
pDestinationControl, BYTE pMatchPosition = FALSE)
ODS 75                         Procedure(String pS, Short pLevel=0),VIRTUAL
ODSD 75                        Procedure(String pS)
PTD 76                         Procedure(String pS, Byte pHideDebug=False),VIRTUAL
RemoveBackSlash 76             Procedure(String pPathOrFile, Byte pTrailing),String
RemoveForwardSlash 77          Procedure(String pPathOrFile, Byte pTrailing)!!,String
SetFileAttrib 78               Procedure(*CString pFile, <Byte pReadOnly>, <Byte
pHidden>, <Byte pSystem>, <Long pAdditionalAttrib>)
SetFileAttrib 78               Procedure(String pFile, <Byte pReadOnly>, <Byte
pHidden>, <Byte pSystem>, <Long pAdditionalAttrib>)
SearchReplace 77               Procedure(String pFind, String pReplace, *CString
pSearchS),Long,PROC
SearchReplace 77               Procedure(String pFind, String pReplace, *String
pSearchS),Long,PROC
SplitFileParts 80              Procedure(String pFileName)
UnixToWindowsPath 80           Procedure(String pUnixPath),String
UrlEncode 81                   Procedure(String pURL),String
UrlDecode 81                   Procedure(String pURL),String
WindowsToUnixPath 81           Procedure(String pWindowsPath),String

Construct 82                   Procedure
Destruct 82                    Procedure
                          End
```

3.4.2 Data Types — Core Class

The core class uses two data type, the FNS_Parts 53, which is used to split up filenames into it's basic parts of drive, path, filename and extension and IT_GUID which is a structure used for GUID.

3.4.2.1 FNS_Parts — Core Class - Data Types

The FNS_Parts is a group that is declared in the ITEquates.inc as:

```
FNS_Parts                 GROUP,TYPE
P_Drive                     CString(IT_MAX_Path)
P_Dir                       CString(IT_MAX_Path)
P_File                      CString(IT_MAX_Path)
P_Ext                       CString(IT_MAX_Path)
                          END
```

The derived FileParts 55 property is used by the SplitFilePart 80 method to store the various file parts.

See also:
FileParts 55
GetFilePart 65
SplitFilePart 80

3.4.2.2 IT_GUID — Core Class - Data Types

The IT_GUID is a group that is declared in the ITWin32Structures.inc as:

```
IT_GUID                   GROUP,TYPE
```

```
Data1                       ULONG
Data2                       ULONG
Data3                       USHORT
Data4                       STRING(8)
                          END
```

This is used by the CreateGUID method

See also:
CreateGUID 60

3.4.3 Properties Core Class

There are currently 11 properties of the Icetips Utility Core class:

```
DebugLevel 55               Byte
EXEName 55                  CString(1025)
FileParts 55                Group(FNS_Parts)
                          End
ProgPath 56                 CString(10241)
ProgramCommandLine 56       CString(10241)
ProgramDebugOn 56           Byte
ComputerName 54             CString(IT_MAX_COMPUTERNAME_LENGTH+1)
UserName 54                 CString(256)
XPThemesPresent 57          Byte
LastAPIError 55             String(255)
LastAPIErrorCode 56         Long
```

3.4.3.1 UserName Core Class - Properties

This property is set automatically by the Constructor and contains the active User Name. You can also set this property and retrieve the user name by using the GetUserName() 69 method.

Example:

```
ITC  ITCoreClass
 Code
 Message('User name: ' & ITC.UserName)
 ITC.GetUserName  !! Set the ITC.UserName property
```

See also:
Construct 82

3.4.3.2 ComputerName Core Class - Properties

This property is set automatically by the Constructor and contains the active Computer Name. You can also set this property and retrieve the user name by using the GetComputerName() 63 method.

Example:

```
ITC  ITCoreClass
 Code
 Message('Computer name: ' & ITC.ComputerName)
```

```
ITC.GetComputerName  !! Set the ComputerName property
```

See also:
Construct 82

3.4.3.3 DebugLevel — Core Class - Properties

DebugLevel is used in the ODS 75 method to determine if the passed string should be sent to OutputDebugString. By default DebugLevel is 0, so to send all strings to OutputDebugString you can use the ODS method with:

Example:

```
ITC  ITCoreClass
 Code
 ITC.ODS('This goes to DebugView',0)
```

or, since the parameter to ODS 75 defaults to 0, you can simply use:

```
ITC  ITCoreClass
 Code
 ITC.ODS('This goes to DebugView')
```

3.4.3.4 EXEName — Core Class - Properties

This is a 1K CString that contains the complete path and name of the exe.

This will contain both the path and the name of the exe.

Example:

```
'C:\Clarion\Apps\MyApp\MyApp.exe'
```

See also:
Construct 82

3.4.3.5 FileParts — Core Class - Properties

This is a GROUP structure derived from FNS_Parts 53. It is used by the SplitFileParts 80 method to store individual parts of a filename.

3.4.3.6 LastApiError — Core Class - Properties

This 256 byte string is set with the last API error message by the GetLastAPIError 66

3.4.3.7 LastApiErrorCode

This LONG properties is set to the last API error code by GetLastAPIError 66 and GetLastAPIErrorCode 67 methods.

3.4.3.8 ProgPath

This is a 10K CString that contains the path to the program.

```
ProgPath                    CString(10241)
```

For example if your program is located in 'C:\Clarion\Apps\MyApps\MyApp.exe', then ProgPath will contain "C:\Clarion\Apps\MyApps"

Note that the path does NOT have a trailing backslash. To fix that use the FixPath method:

Example:

```
ITC ITCoreClass
FP  CSTRING(2049)
 CODE
 FP = ITC.FixPath(ITC.ProgPath))
 MESSAGE('Program Path "' & FP)
```

See also:
Construct 82

3.4.3.9 ProgramCommandLine

This is a 10K CString variable that contains the entire command line for the program.

```
ProgramCommandLine          CString(10241)
```

See also:
Construct 82

3.4.3.10 ProgramDebugOn

This property is set either by passing DEBUG as a runtime parameter to the program, which is then picked up by the Construct method 82, or it can be set by the programmer to True or False. It is used in the ODSD 75 method to determine if the method calls ODS 75 which in turns uses OutputDebugString to print to a debugging viewer.

For the most popular debug viewer visit http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx and download DebugView. It can be set up to run over your network so you can run it on computer A while sending information from an

application running on computer B.

See also:
Construct 82

3.4.3.11 ReplaceString — Core Class - Properties

The ReplaceString property is used in the FindReplace 61 method and is used to contain the string used to do a search and replace on inside the FindReplace 61 method. Since the string is returned by the FindReplace 61 method it needs to be a property and is destroyed in the Destruct 82 method. The ReplaceString is a private property.

It is declared as:

```
ReplaceString               &String,PRIVATE
```

3.4.3.12 UrlStr — Core Class - Properties

Used with the URL encoding functions.

3.4.3.13 XPThemesPresent — Core Class - Properties

This property indicates if the XPThemes template and classes are compiled in the project. This allows certain methods access to the XP Theme options.

See also:
Construct 82

3.4.4 Methods — Core Class

There are currently 34 methods in the Icetips Utility Core class:

```
CreateGUID 60                Procedure(Byte pAddBraces=1),STRING
FixPath 62                   Procedure(*CString pPath),String
FixPath 62                   Procedure(String pPath),String
GetBit 62                    Procedure(Long pVariable, Byte pBitToGet), Byte
GetBitString 155             Procedure(Long pVariable),String
GetComputerName 63           Procedure(),String,PROC         ! Returns the name of
the computer.  Puts it into the ComputerName property
GetFileAttrib 64             Procedure(*CString pFile, <*Byte pReadOnly>, <*Byte
pHidden>, <*Byte pSystem>),Long,PROC
GetFileAttrib 64             Procedure(String pFile, <*Byte pReadOnly>, <*Byte
pHidden>, <*Byte pSystem>),Long,PROC
GetFilePart 65               Procedure(String pFilename,Byte pPart),String
GetLastAPIError 66           Procedure(<*Long pErrorCode>),String
GetLastAPIErrorCode 67       Procedure(),Long
GetTempFilename 68           Procedure(<String pPath>,<String pPrefix>),String
GetTempFolder 69             Procedure(),String
GetFilePart 65               Procedure(String pFilename,Byte pPart),String
```

```
GetUserName 69                  Procedure(),String
IsFileInUse 71                  Procedure(*CString pFile),Byte
IsFileInUse 71                  Procedure(String pFile),Byte
IsFolder 71                     Procedure(*CString pPath),Byte
IsFolder 71                     Procedure(String pPath),Byte
ODS 75                          Procedure(String pS, Short pLevel=0),VIRTUAL
ODSD 75                         Procedure(String pS)
PTD 76                          Procedure(String pS, Byte pHideDebug=False),VIRTUAL
RemoveBackSlash 76              Procedure(String pPathOrFile, Byte pTrailing),String
RemoveForwardSlash 77           Procedure(String pPathOrFile, Byte pTrailing)!!,String
SetBit 78                       Procedure(Long pVariable, Byte pBitToSet, Byte
pBitSet), LONG
SetFileAttrib 78                Procedure(*CString pFile, <Byte pReadOnly>, <Byte
pHidden>, <Byte pSystem>, <Long pAdditionalAttrib>)
SetFileAttrib 78                Procedure(String pFile, <Byte pReadOnly>, <Byte
pHidden>, <Byte pSystem>, <Long pAdditionalAttrib>)
SearchReplace 77                Procedure(String pFind, String pReplace, *CString
pSearchS),Long,PROC
SearchReplace 77                Procedure(String pFind, String pReplace, *String
pSearchS),Long,PROC
SplitFileParts 80               Procedure(String pFileName)
UnixToWindowsPath 80            Procedure(String pUnixPath),String
WindowsToUnixPath 81            Procedure(String pWindowsPath),String

Construct 82                    Procedure
Destruct 82                     Procedure
```

3.4.4.1 AllocateSearchString

Core Class - Methods

Prototype: **(Long pSize),PROTECTED**

pSize Size to allocate.

This method is used internally to allocate memory for the ReplaceString 57 property. This method is protected and should not be called from outside the class.

See also:
ReplaceString 57
FindReplace 61

3.4.4.2 AllocateURLString

Core Class - Methods

Prototype: **(Long pSize),PROTECTED**

pSize New size for the UrlStr 57 property

Re allocates the UrlStr 57 property to be the specified size. This method is protected and should not be called from outside the class.

See also:
UrlDecode 81
UrlEncode 81

3.4.4.3 ByteToHex — Core Class - Methods

Prototype: **(Byte pByte),String**

pByte A byte value to convert to HEX

Returns A string containing the HEX value of pByte

This method takes a byte value and converts it to a two character HEX string

Example:

```
ITC  ITCoreClass
B    Byte
 Code
 B = 65
 Message('The Hex value of ' & B & ' = ' & ITC.ByteToHex(B))
```

See also:
LongToHex 73

3.4.4.4 CountFinds — Core Class - Methods

Prototype: **(String pFind, String pSearchS),Long**

pFind String to search for
pSearchS String to search in

Returns Number of times that pFind is found inside pSearchS

This method is used in FindReplace 61 to determine how many times the string is found. This is used to accurately allocate memroy with AllocateSearchString 58. You can use this method anywhere and it can be very useful if you want to determine how many times a string is found. The search is **not case sensitive**.

Example:

```
ITC  ITCoreClass
 Code
 Message('X is found ' & ITC.CountFinds('X','This x is x'))
 !! Returns 2
```

See also:
FindReplace 61
AllocateSearchString 58

3.4.4.5 CreateGUID

Prototype: **(Byte pAddBraces=1),String**

pAddBraces Adds curly braces around the GUID. This parameter is **true** by default. Example: {87C4B7EF-F219-4FD7-AB94-BEA4287C2E2F} If this parameter is false, no curly braces are added and this GUID would look like 87C4B7EF-F219-4FD7-AB94-BEA4287C2E2F

Returns The GUID formatted as specified by the pAddBraces parameter.

This method uses CoCreateGUID and StringFromGUID2 api calls to create and format a Global Unique Identifier which can be used for all kinds of things. Note that by default the method will add the curly braces around the GUID.

Example:

```
ITC  ITCoreClass
GUID String(40)
 Code
 GUID = ITC.CreateGUID()       !! WITH curly braces
 GUID = ITC.CreateGUID(False) !! WITHOUT curly braces
```

See also:
IT_GUID 53

3.4.4.6 FileExists

Prototype: **(String pFileName, Long pAttribute=FF_:Normal),Long**

pFileName The file name to check if exist. The pFileName parameter can contain wildcards for the filename and extension.

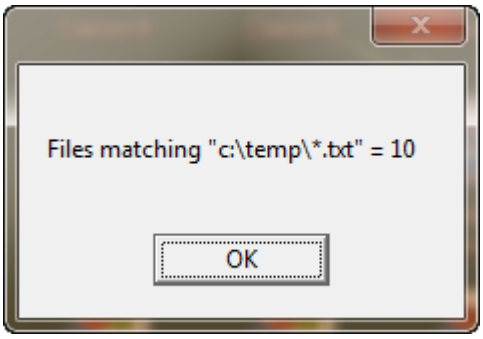
pAttribute Attribute to look for. By default it is set to FF_:Normal. **Added August 3, 2010**

Returns The number of files that matches the pFileName parameter.

This method is to replace the EXISTS function in Clarion. It is not always reliable unless the filename is changed to a ShortPath(). The FileExists method uses the DIRECTORY() function in Clarion to check for the file existence. This has a second benefit also because you can specify wildcards for the filename and the function will return the number of files that exist and match the wildcards.

Example:

```
ITC ITCoreClass
p   CSTRING(2049)
 CODE
 p = 'c:\temp\*.txt'
 MESSAGE('Files matching "' & p & '" = ' & ITC.FileExists(p))
```



See also:
GetFileSize 66

3.4.4.7 FindReplace — Core Class - Methods

Prototype:	**(String pFind, String pReplace, String pSearchS, <*LONG pNewLen>),String**
pFind	The string to search for.
pReplace	The string to replace with.
pSearchS	The string to search and replace in. This can be a string literal or a string variable.
pNewLen	Omittable parameter that receives the new length of the string
Returns	The resulting string with the pFind string replaced with the pReplace string

This method is identical to the SearchReplace 77 method except FindReplace can take a literal string or variable as the string to search in and instead of returning the number of replacements performed, it returns the new string. This makes it a little bit more flexible to use since you don't need to declare a variable to execute the search and replace.

Note that this method will be slightly slower than SearchReplace because it has to count the number of times the pFind is found in pSearchS in order to allocate enough memory for it to start the search and replace process. A simple test with the example code from SearchReplace took 0.71 seconds to run 100,000 (one hundred thousand) iterations of SearchReplace but 1.45 seconds to run the same number of iterations using FindReplace on the same machine. However, for most practical application the difference is not noticable.

Example:
```
S    String(255)
ITC  ITCoreClass
 Code
 S = ITC.FindReplace('now','NOW','Check this out now') ! S is now:  'Check
this out NOW'
```

See also:
SearchReplace 77
AllocateSearchString 58

CountFinds 59
ReplaceString 57

3.4.4.8 FixPath

Prototype: **(String pPath),String**

pPath The path to fix

Returns The fixed path name

This method takes a path name and checks if it contains a trailing backslash or not and returns the path WITH a trailing backslash. It is overloaded with a method that takes a *CString parameter so it can be used with both string variables and CString variables.

Example:

```
ITC ITCoreClass
p  CString(2049)
 Code
 p = 'c:\temp'
 Message('Fixed path = ' & ITC.FixPath(p))
```

See also:
IsFolder 71
RemoveBackslash 76
CheckLeadingBackslash 183
CheckTrailingBackslash 183

3.4.4.9 GetBit

Prototype: **(Long pVariable, Byte pBitToGet), Byte**

pVariable The value to get bit information from

pBitToGet The bit to get the status from. **Note that bits are zero (0) based so the lowest bit is 0, next is 1 and so on.**

Returns Returns the status of the specified bit, either 0 or 1

This method checks the status of a single bit and returns either 0 or 1 depending on if the bit is off or on.

Example:

```
ITC  ITCoreClass
L  Long
S  String(32)
B  Byte
```

```
 Code
 L = 8
 B = 1
 Message('Bit ' & B & ' is: ' & ITC.GetBit(L,B)
```

See also:
GetBitString 155
SetBit 78

3.4.4.10 GetBitString — Core Class - Methods

Prototype: **(Long pVariable),String**

pVariable Value to construct bit string for. This can be any integer variable. Note that this method only accepts 32bit integers at this point.

Returns Returns a bit string with 32 characters.

This method takes any 32bit integer value, positive or negative and returns a 32bit binary string representing the integer value. Note that bits are zero (0) based so the lowest bit is 0, next is 1 and so on.

Example:

```
ITC  ITCoreClass
L  Long
S  String(32)
 Code
 L = 8
 Message('Binary code for ' & L & ' is: ' & ITC.GetBitString(L)
```

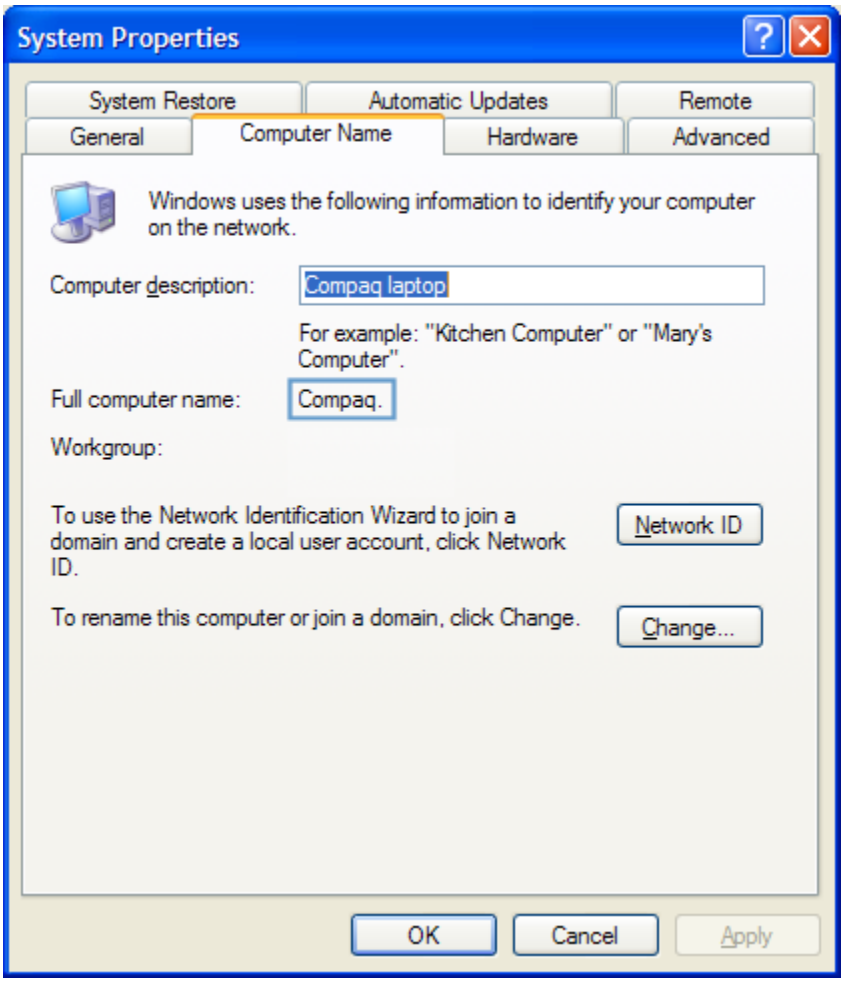
See also:
GetBit 62
SetBit 78

3.4.4.11 GetComputerName — Core Class - Methods

Prototype: **(),String,PROC**

Returns Return the computer name

This method uses the GetComputerName api to retrieve the name of the computer as shown below.



On this computer the GetComputerName would return "Compaq" Note that the APIs normally return the computer name as all upper case.

Example:

```
ITC  ITCoreClass
 Code
 Message('Computer Name: ' & ITC.GetComputerName())
 !! Using the ITC.ComputerName property would also work.
```

See also:

ComputerName 54

3.4.4.12 GetFileAttrib

Core Class - Methods

Prototype:	**(String pFile, <*Byte pReadOnly>, <*Byte pHidden>, <*Byte pSystem>),Long,PROC**
pFile	Name of the file to check
pReadOnly	Optional parameter to receive the Read-Only bit

pHidden	Optional parameter to receive the Hidden bit
pSystem	Optional parameter to receive the System bit
Returns	Returns the file attribute as returned by the GetFileAttributes api.

This method retrieves the file attribute for the specified file, i.e. if it is a read-only, hidden or system file. You can pass in byte variables to receive the information. It is overloaded with a method that takes a *CString parameter so it can be used with both String and CString variables.

Example:

```
ITC  ITCoreClass
f    CString(2049)
ro   Byte
hi   Byte
sy   Byte
 Code
 f = 'c:\temp\myfile.txt'
 ITC.GetFileAttrib(f,ro,hi,sy)
 ! ro, hi and sy will now either be true or false depending on if the
individual attributes are set for the file.
```

See also:
SetFileAttrib 78

3.4.4.13 GetFilePart — Core Class - Methods

Prototype:	**(String pFilename, Byte pPart),String**
pFileName	The name, with or without path, of the file.
pPart	Defines what part of the filename you want returned. The parts available are FNS_Drive, FNS_Path, FNS_File and FNS_Ext. Additional combined parts are FNS_FullPath which returns the drive and path, FNS_FileName which returns the filename and extension and FNS_FullPathFile which returns the drive, path and filename, but not the extension. You can add them up to match any parts you want.
Returns	The part of the filename determined by the pPart.

This method returns the part of a full path filename that you request in the pPart parameter. This is a very useful function when dealing with filenames as it allows you to specify what parts you need. Also note the SplitFileParts 80 method which splits all parts of a filename into the FileParts 55 group.

Example:

```
F    CString(1025)
Fn   CString(1025)
ITC  ITCoreClass
 Code
 F = 'C:\Clarion\Apps\Test\GetFilePart\GetFilePart.app'
 Fn = ITC.GetFilePart(F,FNS_Drive+FNS_Path)
```

```
      ! Returns: 'C:\Clarion\Apps\Test\GetFilePart\'
 Fn = ITU.GetFilePart(F,FNS_Path)
      ! Returns: \Clarion\Apps\Test\GetFilePart\'
 Fn = ITU.GetFilePart(F,FNS_File)
      ! Returns: 'GetFilePart'
 Fn = ITU.GetFilePart(F,FNS_Ext)
      ! Returns: '.app'
 Fn = ITU.GetFilePart(F,FNS_File+FNS_Ext)
      ! Returns: 'GetFilePart.app'
 Fn = ITU.GetFilePart(F,FNS_Drive+FNS_Path+FNS_File)
      ! Returns: 'C:\Clarion\Apps\Test\GetFilePart\GetFilePart'
 Fn = ITU.GetFilePart(F,FNS_Drive+FNS_Path+FNS_File+FNS_Ext)
      ! Returns: 'C:\Clarion\Apps\Test\GetFilePart\GetFilePart.app'
 Fn = ITC.GetFilePart(F,FNS_FullPath)
      ! Returns: 'C:\Clarion\Apps\Test\GetFilePart\'
 Fn = ITC.GetFilePart(F,FNS_FileName)
      ! Returns: 'GetFilePart.app'
 Fn = ITC.GetFilePart(F,FNS_FullPathFile)
      ! Returns: 'C:\Clarion\Apps\Test\GetFilePart\GetFilePart'
```

[****]

See also:

SplitFileParts 80

3.4.4.14 GetFileSize — Core Class - Methods

Prototype: **(String pFilename),Real**

pFileName The name of the file to get the size for

Returns The size of the file.

This method can be used on files that are over 2GB in size.

Example:

```
F    CString(1025)
ITC  ITCoreClass
 Code
 F = 'C:\hiperfil.sys'
 Message('Size of ' & F & ' = ' & ITC.GetFileSize(F))
```

See also:

GetFileAttrib 64
GetFilePart 65

3.4.4.15 GetLastAPIError — Core Class - Methods

Prototype: **(<*Long pErrorCode>,<String pLocation>),String**

[pErrorCode] Optional Parameter that receives the API error code value.

[pLocation] Optional Parameter that has a prefix for the error, such as "Mylocation: (183) Cannot create a file when that file already exists."

Returns Returns a formatted error message from the operating system.

This method checks for an error after calling any Windows API function and returns a formatted error message and optionally an error code. This is very useful method to use when writing API functions to check for errors and get the error message text as the operating system formats it. The function first retrieves the last error from the operating system with the GetLastError api function. It then uses the FormatMessage api to format the error message text and return it to the calling code. The formatting is done with FORMAT_MESSAGE_FROM_SYSTEM + FORMAT_MESSAGE_MAX_WIDTH_MASK

Example:

```
(none)
```

See also:
GetLastApiErrorCode 67
APIErrorHandler 379
LastApiError 55
LastApiErrorCode 56

3.4.4.16 GetLastAPIErrorCode — Core Class - Methods

Prototype: **(),Long**

Returns Returns the last error code from the system.

This method is very useful to check for errors after calling api functions. This method calls the GetLastError api function and returns the result. In fact this works exactly the same as calling GetLastError() api.

Example:

```
(none)
```

See also:
GetLastAPIError 66
APIErrorHandler 379
LastApiError 55
LastApiErrorCode 56

3.4.4.17 GetReplaceString — Core Class - Methods

Prototype: **(),STRING**

Returns Returns the contents of the ReplaceString private property.

This method is used to retrieve the value of the ReplaceString 57 property, which is private and should never be updated outside of the class. This is convenient when calling the FindReplace 61 method with the same data multiple times.

Example:

```
SetCustomerData   PROCEDURE  (CUSTOMER_GROUP pCus)
ITS       ITStringClass
CS        STRING(100)
  CODE
  IF LEN(CLIP(pCus)) > 0
    ITS.ReadFileToString('Template.xml'))

    !! First use the FileString property:
    CS = ITS.FindReplace('##Handle_Number##', pCus.HandleNumber         ,
ITS.FileString)

    !! Then use the ReplaceString by calling GetReplaceString()
    CS = ITS.FindReplace('##Number##',        pCus.Number               ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Group_Handle##',  pCus.GroupHandle          ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Name##',          pCus.Name                 ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Currency##',      pCus.CurrencyHandleCode   ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Email##',         pCus.Email                ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##PhoneNnumber##',  pCus.TelephoneAndFaxnumber,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Website##',       pCus.Website              ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Address##',       pCus.Address              ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##PostalCode##',    pCus.PostalCode           ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##City##',          pCus.City                 ,
ITS.GetReplaceString())
    CS = ITS.FindReplace('##Country##',       pCus.Country              ,
ITS.GetReplaceString())
    RETURN(ITS.GetReplaceString())
  ELSE
    RETURN('')
  END
```

See also:
FindReplace 61

3.4.4.18 GetTempFilename

Core Class - Methods

Prototype:	**(<String pPath>,<String pPrefix>),String**
pPath	Optional path. If omitted the default Temp path will be used
pPrefix	Optional prefix for the temp filename.
Returns	Returns a unique temporary filename.

This method creates a unique filename and returns the full path. If the pPath is specified that path is used, otherwise it will return the temporary path. Please note that this method CREATES the temporary file also. This is done by the GetTempFileName api call to prevent the filename from being

allocated to some other program. If you do not intend to use the file, just need a filename, use Remove() to remove the file.

The filename returned is always a short filename. Use LongPath() to get the Long filename for the returned filename. If you run the app from the Clarion IDE it will pick up a different TEMP folder than if you run it independent of the IDE. That is because the Clarion IDE is 16bit so it has a different enviornment setting than standard 32bit programs.

Example:

```
ITC  ITCoreClass
fn   CString(2049)
 Code
 fn = ITC.GetTempFileName()
 Remove(fn)  ! Remove the file.
 Message('Filename: ' & LongPath(fn))
```

See also:
GetTempFolder 69

3.4.4.19 GetTempFolder — Core Class - Methods

Prototype: **(),String**

Returns Returns the folder for temporary files

This method returns the currently set temp folder. This depends on the environment settings and the %TEMP% environment variable. If you run the app from the Clarion IDE it will pick up a different TEMP folder than if you run it independent of the IDE. That is because the Clarion IDE is 16bit so it has a different enviornment setting than standard 32bit programs.

Example:

```
ITC  ITCoreClass
 Code
 Message('Temp folder: ' & ITC.GetTempFolder())
```

See also:
GetTempFilename 68

3.4.4.20 GetUserName — Core Class - Methods

Prototype: **(),String**

Returns The user name for the currently logged in user.

This method retrieves the username for the currently logged in user and returns it and also sets the UserName 54 property.

Example:

```
ITC  ITCoreClass
 Code
 Message('Current user: ' & ITC.GetUserName())
```

See also:
GetComputerName 63

3.4.4.21 GetXMLDateTime

Prototype:	**(ANY pDate, ANY pTime, <STRING pTimeZone>),STRING**
pDate	The date to include
pTime	The time to include
pTimeZone	Optionally specify time zone identifier. It can be either "Z" for UTC (Greenwich) or +/-HH:MM for time ahead/behind UTC (Greenwich)
Returns	String containing formatted XML date/time

This method takes a date and a time and returns a XML datetime string formatted as "YYYY-MM-DDTHH:MM:SS"

Example:

```
ITC  ITCoreClass
  CODE
  MESSAGE('XML date: ' & ITC.GetXMLDateTime(TODAY(),CLOCK()))
```

See also:
FormatXML 296

3.4.4.22 IsAppframe

Prototype:	**(),Byte**
Returns	True if the current target is an AppFrame window, false if it is not.

This method simply checks if the currently selected target is an AppFrame window or not.

Example:

```
ITC  ITCoreClass
 Code
 If ITC.IsAppframe()
   Message('This is an Appframe window')
 End
```

3.4.4.23 IsFileInUse

Core Class - Methods

Prototype: **(String pFile),Byte**

pFile File name to check if it is locked and in use.

Returns Returns true or false depending on if the file is in use.

This method attempts to open the specified file in GENERIC_WRITE mode to see if it can be opened for writing. If not, it is deemed in use by some other program or process. If the file can be opened the method returns false. A file that is in use can generally be opened for reading. This method is very useful for operations such as deleting files or opening them with write access. If the file is in use it can't be written to. It is overloaded with a method that takes a *CString parameter so it can be used with both string variables and CString variables.

Example:

```
ITC ITCoreClass
Fn  CString(2049)
 Code
 Fn = ITC.EXEName
 ITC.ODS('ExeName = ' & Fn)
 If ITC.IsFileInUse(Fn) !! Should ALWAYS return true
   Message('This program is in use.')
 Else
   Message('This program is not in use (should never happen)')
 End

 Fn = ITC.GetFilePart(ITC.EXEName,FNS_FullPath+FNS_File) & '.app'
 ITC.ODS('AppName = ' & Fn)
 If ITC.IsFileInUse(Fn) !! Should not be in use.
   Message('This appfile is in use.')
 Else
   Message('The appfile is not in use.')
 End
```

See also:
IsFolder 71

3.4.4.24 IsFolder

Core Class - Methods

Prototype: **(String pPath),String**

pPath The path to check if it's a folder or a filename

Returns Returns TRUE if the pPath is a folder.

This method returns true if the path name passed in is a folder and false if it is a file. You can easily have path named C:\myfile.txt as well as a file named C:\myfile.txt. The IsFolder method will determine if it is a folder or not and return the appropriate value.

It is overloaded with a method that takes a *CString parameter so it can be used with both string variables and CString variables.

Example:

```
ITC ITCoreClass
Fn  CString(2049)
S   CString(1024)
 Code
 Fn = ITC.EXEName
 S = '"' & Fn & '"' & Choose(ITC.IsFolder(Fn)=True,' IS ',' is NOT ') & 'a
folder.'
 Fn = LongPath()
 S = S & '|' & '"' & Fn & '"' & Choose(ITC.IsFolder(Fn)=True,' IS ',' is
NOT ') & 'a folder.'
 Message(S,'IsFolder')
```

See also:
IsFileInUse 71
GetFileAttrib 64

3.4.4.25 IsFolderWritable — Core Class - Methods

Prototype: **(String pPath),Byte**

pPath The path to check if it can be written to.

Returns Returns either TRUE if the folder can be written to or FALSE if it cannot be written to.

This method can be used to test if a specific folder can be written to or not. If it cannot the error information are available in the LastAPIError 55 and LastAPIErrorCode 56 properties immediately after calling the method. The way the method works is that it attempts to create a temporary file in the folder. The filename is made up of a tilde character, a GUID and has a .tmp extension. An example: ~4944E145-1D36-401B-AD01-922685E0CA90.tmp

Example:

```
ITC  ITCoreClass
PFN  CSTRING(1025)
  CODE
  PFN = 'C:\temp'
  IF NOT ITC.IsFolderWritable(PFN)
    MESSAGE('The project folder cannot be written to "' &|
            ITC.GetFilePart(PFN, FNS_FullPath) & '"', |
            'ERROR:  Cannot write to this folder')
  END
```

See also:
IsFolder 71
IsFileInUse 71

GetLastAPIError 66

3.4.4.26 LongToHex — Core Class - Methods

Prototype: **(Long pDecValue),String**

pDecValue Decimal value to get Hexadecimal value from

Returns The hexadecimal value of pDecValue

This method uses the internal sPrintf function to format the decimal value to a hexadecimal string type with 8 characters.

Example:

```
D    Long
H    String(10)
ITU  ITUtilityClass
 Code
 D = 123456
 H = ITU.LongToHex(L)  ! Returns 3039
```

3.4.4.27 Lesser — Core Class - Methods

Prototype: **(? pA, ? pB), ?**

pA Value to compare to pB

pB Value to compare to pA

Returns The lesser value

This method can be used to find the lower/lesser value of two.

Example:

```
ITC  ITCoreClass
A  Long
B  Long
  Code
  A = 1
  B = 2
  MESSAGE('The lesser value of A/B is: ' & ITC.Lesser(A,B))
  !! Will show 1 (A)
```

See also:

Greater 74

3.4.4.28 MatchControlSize

Prototype: **(LONG pSourceControl, LONG pDestinationControl, BYTE pMatchPosition = FALSE)**

pSourceControl	Control to copy size from
pDestinationControl	Control to copy size to
pMatchPosition	Indicates if the destination control should be moved to the position of the source control. Defaults to FALSE.

This method sets the size and optionally the position of the destination control to match the size and position of the source control. Comes in handy when you need to stack controls at runtime but would like to keep them separated at design time.

Example:

```
Window WINDOW('MatchControlSize'),AT(,,257,98)
    BOX,AT(14,8,47,40),USE(?BOX1),COLOR(COLOR:Black),FILL(COLOR:Black),
LINEWIDTH(1)
    REGION,AT(85,25),USE(?REGION1)
  END
ITC  ITCoreClass
  CODE
  OPEN(Window)
  ITC.MatchControlSize(?BOX1,?REGION1,TRUE)
  ACCEPT
  END
  CLOSE(Window)
```

See also:

3.4.4.29 Greater

Prototype: **(? pA, ? pB), ?**

pA	Value to compare to pB
pB	Value to compare to pA
Returns	The greater value

This method can be used to find the lower value of two.

Example:

```
ITC  ITCoreClass
A  Long
B  Long
  Code
  A = 1
  B = 2
```

```
MESSAGE('The lesser value of A/B is: ' & ITC.Greater(A,B))
!! Will show 1 (B)
```

See also:
Lesser 73

3.4.4.30 Message — Core Class - Methods

Under construction

3.4.4.31 ODS — Core Class - Methods

Prototype: **(String pS, Short pLevel=0), VIRTUAL**

pS String to send to OutputDebugString
pLevel Indicates DebugLevel 55 to compare with. This parameter defaults to 0.

This method sends the string directly to OutputDebugString after converting it to CString if the pLevel is equal or more than the DebugLevel 55 property value.

Example:

```
ITC  ITCoreClass
 Code
 ITC.ODS('Check if this shows up in DebugView')
 ITC.DebugLevel=2
 ITC.ODS('This should NOT show up',1)
```

See also:
DebugLevel 55

3.4.4.32 ODSD — Core Class - Methods

Prototype: **(String pS)**

pS String to send to OutputDebugString

This method sends the string to ODS 75 if the ProgramDebugOn 56 property is set to True.

Example:

```
ITC  ITCoreClass
 Code
 ITC.ProgramDebugOn = True
 ITC.ODSD('This will show up in DebugView')
  ITC.ProgramDebugOn = False
  ITC.ODSD('This will NOT show up in DebugView')
```

See also:
ProgramDebugOn 56

ODS 75

3.4.4.33 PTD

Core Class - Methods

Prototype: **(String pS, Byte pHideDebug=False), VIRTUAL**

pS String to send to OutputDebugString

pHideDebug Flag that can be used in derived methods to prevent the method to send the output to OutputDebugString, but rather redirect it to some other output device, such as a file.

This virtual method is used to **P**rint **T**o **D**ebug and send the output to tools such as DebugView from www.systeminternals.com. Microsoft acquired SystemInternals in July 2006, but the utilities are still free and available for download at http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx Also check out DebugView++ This method calls the ODS 75 method passing the pS parameter to it if the pHideDebug is false.

Example:

```
ITC  ITCoreClass
 Code
 ITC.PTD('Check if this shows up in DebugView')
 ITC.PTD('This should not show up in DebugView',True)
```

See also:
ODS 75

3.4.4.34 RemoveBackSlash

Core Class - Methods

Prototype: **(String pPathOrFile, Byte pTrailing),String**

pPathOrFile Path or filename to check.

pTrailing If true, the method strips trailing backslashes, if false it strips leading backslashes.

Returns The stripped path or filename.

This function will remove either leading or trailing backslashes from file/path names.

Example:

```
Fn   CString(1025)
ITC  ITCoreClass
 Code
 Fn = 'C:\Clarion\'
 Fn = ITC.RemoveBackSlash(Fn,True)   ! Fn is now 'C:\Clarion'
 Fn = '\Clarion\'
 Fn = ITC.RemoveBackSlash(Fn,False)  ! Fn is now 'Clarion\'
 Fn = ITC.RemoveBackSlash(Fn,True)   ! Fn is now 'Clarion'
```

See also:
RemoveForwardSlash 77

3.4.4.35 RemoveForwardSlash — Core Class - Methods

Prototype: **(String pPathOrFile, Byte pTrailing),String**

pPathOrFile URL, path or filename to check.

pTrailing If true, the method strips trailing forwardslashes, if false it strips leading forwardslashes.

Returns The stripped path or filename.

This function will remove either leading or trailing forwardslashes from file/path names.

Example:

```
URL  CString(1025)
ITC  ITCoreClass
 Code
 Fn = 'http://www.icetips.com/'
 Fn = ITC.RemoveForwardSlash(Fn,True)    ! Fn is now
'http://www.icetips.com'
 Fn = '//www.icetips.com/'
 Fn = ITC.RemoveForwardSlash(Fn,False)   ! Fn is now 'www.icetips.com/'
 Fn = ITC.RemoveForwardSlash(Fn,True)    ! Fn is now //www.icetips.com'
```

See also:
RemoveBackSlash 76

3.4.4.36 SearchReplace — Core Class - Methods

Prototype: **(String pFind, String pReplace, *String pSearchS),Long,PROC**

pFind The string to search for.

pReplace The string to replace with.

pSearchS The string to search and replace in. Note that this is passed by address so you must pass a variable to this method.

Returns The replaced string.

This is a simple but powerful search and replace method. The method is overloaded with a method that takes a *CString parameter so it can be used with both string variables and CString variables. The search is NOT case sensitive.

NOTE: The string or cstring variable used, is NOT dynamic and is NOT expanded to hold the data if it is too big. If you anticipate to replace a small string with a big one, you have to consider that the

variable that you are passing in with the string (pSearchS) may not be big enough to hold the string after the search and replace has taken place.

Example:

```
S    String(255)
ITC  ITCoreClass
 Code
 S = 'Check this out now'
 ITC.SearchReplace('now','NOW',S) ! S is now:  'Check this out NOW'
```

See also:

Construct 82

3.4.4.37 SetBit

Core Class - Methods

Prototype:	**(Long pVariable, Byte pBitToSet, Byte pBitSet), LONG**
pVariable	The value to set bit information in
pBitToSet	The bit to change status. **Note that bits are zero (0) based so the lowest bit is 0, next is 1 and so on.**
pBitSet	Either 0 or 1 depending on if you want to turn the bit off or on.

This method is used to set a single bit in an integer value.

Example:

```
ITC  ITCoreClass
L  Long
S  String(32)
B  Byte
 Code
 L = 8
 B = 1
 L = ITC.SetBit(L,B,0)
 Message('Bit ' & B & ' is now: ' & ITC.GetBit(L,B)
```

See also:

GetBit 62
GetBitString 155

3.4.4.38 SetFileAttrib

Core Class - Methods

Prototype:	**(String pFile, <Byte pReadOnly>, <Byte pHidden>, <Byte pSystem>, <Long pAdditionalAttrib>),IT_DWORD,PROC**
pFile	The filename to change attributes on.
pReadOnly	Set the Read-Only attribute
pHidden	Set the Hidden attribute
pSystem	Set the System attribute

pAdditionalAttrib Set additional attributes.

Returns Returns 0 if the function failed and non-zero value if it succeeded.

This method changes the attributes of the specified file. If only the filename is specified or the pReadOnly, pHidden and pSystem are all set to zero the attribute is set to FILE_ATTRIBUTE_ARCHIVE. The example below is from the CoreClassDemo.app and demonstrates the use of variables to set the attributes. The demo app also shows how to retrieve the attributes when a file is selected and set the variables. We suggest you study the code in the demo application.

Example:

```
ITC ITCoreClass
S   CString(256)
 Code
 If Not Loc:AttribFile
   Post(EVENT:Accepted, ?LookupFile)
   Exit
 End
 If Loc:ReadOnly+Loc:Hidden+Loc:System > 0
   If Loc:ReadOnly
     S = ' +ReadOnly'
   End
   If Loc:Hidden
     S = S & ' +Hidden'
   End
   If Loc:System
     S = S & ' +System'
   End
   S = 'to' & S
 Else
   S = ' back to Archive Only'
 End

 If Message('Are you sure that you want to change the attributes for "' &
Clip(Loc:AttribFile) &|
          '" ' & S &|
          '?||Note that you may need to change your Windows Explorer
settings to see Hidden and System files!!!',|
          'SetFileAttrib',|
          ICON:Question,BUTTON:Yes+BUTTON:No,BUTTON:No) = BUTTON:Yes
   If Not ITC.SetFileAttrib(Loc:AttribFile,Loc:ReadOnly,Loc:Hidden,Loc:
System)
     Message('Could not set the attributes on the file:||' & ITW.
GetLastAPIError())
   End
 End
```

See also:
GetFileAttrib 64

3.4.4.39 SplitFileParts

Prototype: **(String pFileName)**

pFileName Name of the file to split up

This method splits up a filename that is passed to it into drive, directory, filename and extension. These parts are stored in the FileParts group derived from FNS_Parts 53. The method does not return any data, instead access the group components directly, see below. The FileParts group is cleared on each call to SplitFileParts so you can not rely on information from previous call to be available after a second call to SplitFileParts.

Example:

```
ITC  ITCoreClass
S  String(1024)
 Code
 S = 'C:\Clarion\Apps\MyApp\MyApp.exe'
 ITC.SplitFileParts(S)
 Message('File parts: ' &|
         'Drive: ' & ITC.FileParts.P_Drive &|
         '|Dir:  ' & ITC.FileParts.P_Dir &|
         '|File: ' & ITC.FileParts.P_File &|
         '|Ext:  ' & ITC.FileParts.P_Ext)
```

See also:
GetFilePart 65

3.4.4.40 TranslateIconString

Under construction

3.4.4.41 UnixToWindowsPath

Prototype: **(String pUnixPath),String**

pUnixPath Path with / separated directory or folder names.

Returns Path with \ separated directory or folder names.

This method is useful when duplicating paths on a local machine and on a server.

Example:

```
ITC  ITCoreClass
ServerPath  CString(256)
LocalPath   CString(256)
 Code
 ! Path() returns 'C:\Clarion\Apps\MyApp'
 ServerPath = '/images/thisimage.png'
 LocalPath  = Path() & ITC.UnixToWindowsPath(ServerPath)
 ! LocalPath is now: 'C:\Clarion\Apps\MyApp\images\thisimage.png'
```

See also:
WindowsToUnixPath 81

3.4.4.42 UrlDecode — Core Class - Methods

Under Construction

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.4.4.43 UrlEncode — Core Class - Methods

Under Construction

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.4.4.44 WindowsToUnixPath — Core Class - Methods

Prototype:	**(String pWindowsPath),String**
pWindowsPath	Path with \ separated directory or folder names.

Returns Path with / separated directory or folder names.

This method is useful when duplicating paths on a local machine and on a server.

Example:

```
ITC  ITCoreClass
ServerPath  CString(256)
LocalPath   CString(256)
 Code
 LocalPath  = '\images\thisimage.png'
 ServerPath  = ITC.WindowsToUnixPath(LocalPath)
 ! ServerPath is now: '/images/thisimage.png'
```

See also:
UnixToWindowsPath 80

3.4.4.45 Construct

Core Class - Methods

Prototype: **None**

The Core class Construct method retrieves information about the running program and stores them in the class properties:

EXEName 55	Set to COMMAND('0') which contains the full path to the executable name that is running.
ProgPath 56	This stores the drive + path of the EXEName
ProgramCommandLine 56	This stores the whole command line of the program as returned by COMMAND('')
ProgramDebugOn 56	This stores the value of a runtime parameter as returned by COMMAND('DEBUG')
XPThemesPresent 57	This indicates if the XP Themes are present or not.
ComputerName 54	This stores the computer name.
UserName 54	This stores the user name of the currently logged in user.

See also:
EXEName 55
ProgPath 56
ProgramCommandLine 56
ProgramDebugOn 56

3.4.4.46 Destruct

Core Class - Methods

Prototype: **None**

The Core class Destruct method currently has no code in it.

Core Class *(ITCoreClass)*

The Core Class contains 34 methods as of version 1.1.2397 that was released on May 4, 2011. The Core class is what all the other classes are based on so it has some very basic methods that are used all over the place in the other classes. Note that not all of the methods are documented as of that release.

Below are short tutorials on how to accomplish various things with this class. The tutorials are short and demonstrate simple use of the methods.

Getting and setting file information

When dealing with files it is often desirable to be able to know, for example, if a file has the read-only attribute set. In some cases it can also be useful to be able to change the attributes of files, for example to change a hidden file to be not hidden or to mark a file to be read-only.

The Core class has several methods to get information about files and folders. To start with, let's take a look at the GetFileAttrib 64 and SetFileAttrib 78.

Example:

```
ITC  ITCoreClassClass
RO   Byte
HI   Byte
SY   Byte
 Code
 ITC.GetFileAttrib('C:\Temp\Filename.txt',RO,HI,SY)
 Message('Read-Only = ' & RO & '|Hidden = ' & HI & '|System File = ' & SY)
```

This will give us the attributes of this file. The GetFileAttrib 64 and SetFileAttrib 78 only retrieve and set the Read-only, Hidden and System attributes.

Example:

```
ITC  ITCoreClassClass
RO   Byte
HI   Byte
SY   Byte
 Code
 RO = True
 HI = False
 SY = False
 ITC.SetFileAttrib('C:\Temp\Filename.txt',RO,HI,SY)  !! Set the attributes

 !! Set it to false to make sure we are getting the information from the file.
 RO = False
 ITC.GetFileAttrib('C:\Temp\Filename.txt',RO,HI,SY)  !! Get the attributes
 Message('Read-Only = ' & RO & '|Hidden = ' & HI & '|System File = ' & SY)
```

Now you should see that the Read-only has been set on the file. Take a look in Windows Explorer to make sure.

There are two other functions that give us information about files. IsFolder 71 and IsFileInUse 71 will let us know if a file name is a folder or a file and IsFileInUse 71 will tell us if a specified file can be written to or not.

When dealing with foldernames it is sometimes very important to be able to distinguish if a file name is a foldername or not. With valid foldernames such as MyFile.TXT it could lead to all sorts of problems if we passed this into a method or function that attempted to write to the file. In this case the file name is

perfectly valid for both a folder and a file. This is where IsFolder 71 comes to the rescue. It will ONLY return True if the file name passed to it is a name of a folder by getting the attribute of the file with the GetFileAttributes api call.

Example:

```
ITC  ITCoreClass
FN   CString(2049)
 Code
 FN = 'C:\temp\myfile.txt'
 If Not ITC.IsFolder(Fn)
   Remove(Fn)
 End
```

In this case the file would only be removed if it is a regular file, not if it is a folder. This can be very useful when you are writing generic methods that can take a parameter with a file name or a foldername and need to work differently depening on if the file name is actually a name of a folder or a file.

Get temporary file names and temporary folder

Sometimes you need to place some data temporarily into a file and then remove the file. Rather than create those files in the application folder, which can cause virtualization to kick in, or in your data folder, it can be very convenient to write those files to the temporary folder. The GetTempFilename 68 and GetTempFolder 69 methods are designed to take care of that.

GetTempFilename 68 creates a temporary file name AND creates the temporary file in either the specified folder or in the temporary folder - same as returned by GetTempFolder 69.

Example:

```
ITC  ITCoreClass
Fn   CString(2049)
Pref CString(4)
 Code
 Fn = ITC.GetTempFileName()
 Message('Temp filename: ' & Fn)
```

This will return a file name in the temp folder and at this point this file already exists. In this case the function will return a unique file name that is usually build up with hexadecimal characters and could look something like C:\WINDOWS\TEMP\35.tmp

Note that the GetTempFileName 68() always returns the ShortPath() of the path.

Example:

```
ITC  ITCoreClass
Fn   CString(2049)
Pref CString(4)
 Code
 Pref = 'ITCTEST'
 Fn = ITC.GetTempFilename(Path(),Pref)
 Message('Temp filename: ' & Fn)
```

In this case the function will return (and create) a file called something like ITC2F.tmp in the currently active folder. The Pref variable is an optional prefix that you can specify but note that only the first 3 characters of the prefix are used. This is a limitation of the API call, not of the GetTempFilename 68 method! This would return something like C:\Clarion\Apps\C63\Products\UTILIT~1\Demos\CORECL~1\ITC36.tmp

Example:

```
ITC  ITCoreClass
```

```
Fn   CString(2049)
 Code
 Fn = ITC.GetTempFilename(Path(),Pref)
 Remove(Fn)
 Fn = ITC.GetFilepart(Fn,FNS_FileName)
 Fn = '"' & LongPath() & '\ITCTEST' & Fn & '"'
 Message('Temp filename: ' & Fn)
```

Of course there is nothing preventing you from being creative and creating a longer prefix to your temporary file name! The above code creates the temp file name, then removes the file that it created. It then retrieves the file name only (file name + extension) from the full pathname returned by GetTempFilename 68 and adds "ITCTEST" in front of the file name as it combines it with the Path(). This would return something like "C:\Clarion\Apps\C63\Products\UtilityClass\Demos\CoreClass\ITCTEST34.tmp" Notice that in this case we use LongPath() instead of Path() to get the long path to the file. Also notice that we remove the file created by the GetTempFilename 68 right after we call it before we start manipulating the file name.

3.5 Date Class

3.5.1 Overview Date Class

The Date Class contains methods that deal with dates, day names, week numbers, month names etc.

```
ITDateClass
Class(ITWindowsClass),TYPE,Module('ITDateClass.clw'),Link('ITDateClass',_ITUtilLink
Mode_),DLL(_ITUtilDllMode_)

Months 88                       &IT_MonthQueue
Days 87                         &IT_WeekDayQueue
Q1 88                           Long,Dim(4)
Q2 88                           Long,Dim(4)
DE 87                           Long,Dim(4)

InitMonthNames 120              Procedure
InitDayNames 120                Procedure
SetMonthName 121                Procedure(Byte pMonth, String pMonthName, <String
pShortName>)
GetMonthName 100                Procedure(Byte pMonth, Byte pLongName=True),String
SetDayName 121                  Procedure(Byte pDay, String pDayName, <String
pShortName>)
GetDayName 92                   Procedure(Byte pDay, Byte pLongName=True),String
GetPreviousWeekDay 108          Procedure(Long pBaseDate, Byte pWeekDay, Byte
pDayIfSame=True),LONG
GetNextWeekDay 105              Procedure(Long pBaseDate, Byte pWeekDay, Byte
pDayIfSame=True),LONG
GetWeekNumber 117               Procedure(Date pDate),Byte  !! ISO 8601

GetThisWeek 113                 Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetLastWeek 96                  Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetNextWeek 104                 Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetThisWorkWeek 114             Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetLastWorkWeek 97              Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetNextWorkWeek 106             Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetThisMonth 111                Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetLastMonth 94                 Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetNextMonth 102                Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetThisQuarter 112              Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetLastQuarter 95               Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetNextQuarter 103              Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetThisYear 115                 Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetLastYear 98                  Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetNextYear 107                 Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
```

```
GetLast12Months 92          Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetMonthToDate 101          Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetQuarterToDate 110        Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetYearToDate 119           Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetMonthFromDate 99         Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetQuarterFromDate 109      Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC
GetYearFromDate 118         Procedure(Date pDate, *Date pFromDate, *Date
pToDate),LONG,PROC

SetWeekStartDay 122         Procedure(Byte pStartDay)
GetWeekStartDay 117         Procedure(),Byte
GetWeekFirstDay 116         Procedure(Date pStartDate, Long pWeek),Long

DateDiff 90                 Procedure(Byte pDatePart=1, Date pStartDate, Date
pEndDate),Long
DateAdd 90                  Procedure(Byte pDatePart=1, Long pNumber, Date
pStartDate),Long
GetDate 91                  Procedure(Date pDate, Long pPeriods, Byte
pDatePart=1),Long

Construct 122               Procedure
Destruct 122                Procedure
                        End
```

3.5.2 Properties — Date Class

There are currently 6 properties in the Date Class:

```
Months 88                   &IT_MonthQueue
Days 87                     &IT_WeekDayQueue
Q1 88                       Long,Dim(4)
Q2 88                       Long,Dim(4)
DE 87                       Long,Dim(4)
WeekStartDay 88             Byte  !! IT_Sunday - IT_Saturday
```

3.5.2.1 Days — Date Class - Properties

The Days property is used in the InitDayNames 120, SetDayName 121, GetDayName 92 and Construct 122 methods and is used to store information about weekdays, such as names, week day number (used for modulus calculations) etc.

It is declared as:

```
Days                        &IT_WeekDayQueue
```

3.5.2.2 DE — Date Class - Properties

The DE property is a dimensioned property used in the GetThisQuarter 112, GetLastQuarter 95, GetNextQuarter 103, GetQuarterToDate 110, GetQuarterFromDate 109 and Construct 122 methods and

is used to store the end date for each of the 4 quarters, i.e. 31, 30, 30 and 31.

It is declared as:

```
DE                          Long,Dim(4)
```

3.5.2.3 Months

The Months property is used in the InitMonthNames 120, SetMonthName 121, GetMonthName 100 and Construct 122 methods and is used to store the month names, both short (abbreviated) and long (full) as well as the month number.

It is declared as:

```
Months                      &IT_MonthQueue
```

3.5.2.4 Q1

The Q1 property is a dimensioned property and is used in the GetThisQuarter 112, GetLastQuarter 95, GetNextQuarter 103, GetQuarterToDate 110, GetQuarterFromDate 109 and Construct 122 methods. It is used to store the start month of each of the 4 quarters, i.e. 1, 4, 7 and 10.

It is declared as:

```
Q1                          Long,Dim(4)
```

3.5.2.5 Q2

The Q2 property is a dimensioned property and is used in the GetThisQuarter 112, GetLastQuarter 95, GetNextQuarter 103, GetQuarterToDate 110, GetQuarterFromDate 109 and Construct 122 methods. It is used to store the start month of each of the 4 quarters, i.e. 1, 4, 7 and 10.

It is declared as:

```
Q2                          Long,Dim(4)
```

3.5.2.6 WeekStartDay

The WeekStartDay property is used in the SetWeekStartDay 122, GetWeekStartDay 117, GetThisWeek 113, GetLastWeek 96, and GetNextWeek 104 methods and is used to store the day indicating the first day of the week. The SetWeekStartDay 122 is called from the Construct 122 method to set the week starting day according to ISO 8601 from 1988 to Monday. It can be changed at any time by using the SetWeekStartDay 122 and retrieved by the GetWeekStartDay 117 methods.

It is declared as:

```
WeekStartDay                Byte
```

3.5.3 Methods Date Class

There are currently 39 methods in the Date Class.

```
GetDayName 92                   Procedure(Byte pDay, Byte pLongName=True),String
GetMonthName 100                Procedure(Byte pMonth, Byte pLongName=True),String
GetNextWeekDay 105              Procedure(Long pBaseDate, Byte pWeekDay, Byte
pDayIfSame=True),LONG
GetPreviousWeekDay 108          Procedure(Long pBaseDate, Byte pWeekDay, Byte
pDayIfSame=True),LONG
InitDayNames 120                Procedure
InitMonthNames 120              Procedure
SetDayName 121                  Procedure(Byte pDay, String pDayName, <String
pShortName>)
SetMonthName 121                Procedure(Byte pMonth, String pMonthName, <String
pShortName>)
GetWeekNumber 117               Procedure(Date pDate),Byte  !! ISO 8601

SetWeekStartDay 122             Procedure(Byte pStartDay)
GetWeekStartDay 117             Procedure(),Byte
GetWeekFirstDay 116             Procedure(Date pStartDate, Long pWeek),Long

DateDiff 90                     Procedure(Byte pDatePart=1, Date pStartDate, Date
pEndDate),Long
DateAdd 90                      Procedure(Byte pDatePart=1, Long pNumber, Date
pStartDate),Long
GetDate 91                      Procedure(Date pDate, Long pPeriods, Byte
pDatePart=1),Long

GetThisWeek 113                 Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLastWeek 96                  Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetNextWeek 104                 Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetThisWorkWeek 114             Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLastWorkWeek 97              Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetNextWorkWeek 106             Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetThisMonth 111                Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLastMonth 94                 Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetNextMonth 102                Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetThisQuarter 112              Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLastQuarter 95               Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetNextQuarter 103              Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetThisYear 115                 Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLastYear 98                  Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetNextYear 107                 Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetLast12Months 92              Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
```

```
GetMonthToDate 101          Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetQuarterToDate 110        Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetYearToDate 119           Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetMonthFromDate 99         Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetQuarterFromDate 109      Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC
GetYearFromDate 118         Procedure(Date pDate, *ANY pFromDate, *ANY
pToDate),LONG,PROC

Construct 122               Procedure
Destruct 122                Procedure
```

3.5.3.1 DateAdd

Prototype: **(Byte pDatePart=1, Long pNumber, Date pStartDate),Long**

pDatePart Indicates what type of unit to use. Can be IT_Days, IT_Weeks, IT_Months or IT_Years. Defaults to IT_Days.

pNumber Number of units to add. This number can be positive or negative

pStartDate Base date for the calculations.

Returns The date calculated by adding pNumber of pDatePart units to the pStartDate.

This method adds or subtracts a set number of units from the base date. The unit can be IT_Days, IT_Weeks, IT_Months or IT_Years.

Example:

```
ITD  ITDateClass
FD   Date
 Code
 FD = Date(7,1,2010)
 ITD.ODS('DateAdd = ' & Format(ITD.DateAdd(IT_Months, -3,FD),@D18))  !! April 1,
2010
 ITD.ODS('DateAdd = ' & Format(ITD.DateAdd(IT_Months,  3,FD),@D18))  !! October 1,
2010
 ITD.ODS('DateAdd = ' & Format(ITD.DateAdd(IT_Months,-10,FD),@D18))  !! September
1, 2009
 ITD.ODS('DateAdd = ' & Format(ITD.DateAdd(IT_Months,-24,FD),@D18))  !! July 1,
2008
```

See also:

DateDiff 90

GetDate 91

3.5.3.2 DateDiff

Prototype: **(Byte pDatePart=1, Date pStartDate, Date pEndDate),Long**

pDatePart Indicates what type of unit to use. Can be IT_Days, IT_Weeks, IT_Months or

	IT_Years. Defaults to IT_Days
pStartDate	The start date to compare to the pEndDate parameter.
pEndDate	The end date to compare to the pStartDate parameter.

Returns The number of boundaries crossed between the two specified dates.

This method is compatible with the DATEDIFF function in MSSQL and several other SQL dialects. It takes two dates and counts the boundaries for the specified unit between the two dates. For example if you use Monday last week to Tuesday this week and use IT_Weeks as unit, it will return 1 week. If you use July 31st as pStartDate and August 1st as pEndDate and IT_Month as unit the function will return 1 as the boundaries between months have been crossed once. Same if you use July 1st and August 31, it will still return only 1 as only one month boundary has been crossed. Note that MSSQL uses Sunday as week start day so to get a 100% MSSQL compatible DateDiff you need to use SetWeekStartDay 122(IT_Sunday) before you use DateDiff 90 or DateAdd 90.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 FD = Date(7,1,2010)
 TD = Date(8,24,2011)
 ITD.SetWeekStartDay(IT_Sunday)

 ITD.ODS('DateDiff Days   FD: ' & Format(FD,@d18) & ', TD: ' &|
         Format(TD,@d18) & ' = ' & ITD.DateDiff(IT_Days,FD,TD))   !! 409 days
 ITD.ODS('DateDiff Weeks  FD: ' & Format(FD,@d18) & ', TD: ' &|
         Format(TD,@d18) & ' = ' & ITD.DateDiff(IT_Weeks,FD,TD))  !! 60 weeks
 ITD.ODS('DateDiff Months FD: ' & Format(FD,@d18) & ', TD: ' &|
         Format(TD,@d18) & ' = ' & ITD.DateDiff(IT_Months,FD,TD)) !! 13 months
```

See also:
SetWeekStartDay 122

DateAdd 90

GetDate 91

3.5.3.3 GetDate — Date Class - Methods

Prototype: **(Date pDate, Long pPeriods, Byte pDatePart=1),Long**

pDate	Base date for the calculations.
pPeriods	Number of period units to add. This number can be positive or negative
pDatePart	Indicates what type of unit to use. Can be IT_Days, IT_Weeks, IT_Months or IT_Years. Defaults to IT_Days.

Returns The date calculated by adding pPeriods of pDatePart units to the pDate.

This method adds - or subtracts - a specified number of period units from the pDate base date. This method is basically the same as the DateAdd 90 method, but different parameter order.

Example:

```
ITD  ITDateClass
FD   Date
 Code
 FD = Date(7,1,2010)

 ITD.ODS('GetDate = ' & Format(ITD.GetDate(FD, -3, IT_Months),@D18))  !! Apr 1,
2010
 ITD.ODS('GetDate = ' & Format(ITD.GetDate(FD,  3, IT_Months),@D18))  !! Oct 1,
2010
 ITD.ODS('GetDate = ' & Format(ITD.GetDate(FD,-10, IT_Months),@D18))  !! Sep 1,
2009
```

See also:
DateAdd 90

3.5.3.4 GetDayName

Prototype: **(Byte pDay, Byte pLongName=True),String**

pDay Day of the week, starting with 1 for Sunday

pLongName Determine if it should return the long/full day name. This parameter is true by default.

Returns The day name as it has been specified by the Constructor or by updating the day names by SetDayName

This method is used to retrieve the day name, such as "Sunday" based on a day number, 0 for Sunday, 1 for Monday and so on. The Constructor 122 calls the InitDayNames 120 which sets the English day names, both long and short (3 character abbreviated day names, Sun, Mon, etc.) Day name constants are declared in the ITEquates.inc file to make day selections easy, IT_Sunday, IT_Monday, IT_Tuesday, IT_Wednesday, IT_Thursday, IT_Friday, IT_Saturday.

Example:

```
ITD  ITDateClass
TheDate  Date
 Code
 TheDate = Date(1,1,2012)
 ITD.GetDayName(TheDate % 7, True)  !! Get the day name for Jan 1, 2012, which
should return "Sunday"
 Message('January 1, 2012 was a: ' & ITD.GetDayName(TheDate % 7))
```

See also:
SetDayName 121

InitDayNames 120

Construct 122

3.5.3.5 GetLast12Months

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.

Returns Number of days in the period

This method returns the start and end dates for a period that started 12 months prior to the pDate. The pToDate will be equal to the pDate..

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLast12Months(D,FD,TD)
 ITD.ODS('GetLast12Months      Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.6 GetLastMonth

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start date and end date of the last/previous month based on the pDate base date. Leap years do not affect this method and neither do changes in years, i.e. if pDate is in January GetLastMonth will return the first and last dates for December 1 and 31 the year before.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLastMonth(D,FD,TD)
 ITD.ODS('GetLastMonth         Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119

GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.7 GetLastQuarter — Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the first and last date in the last/previous quarter based on the pDate base date. If the current quarter is the first quarter, this method will return the dates for October 1 and December 31, the year before.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLastQuarter(D,FD,TD)
 ITD.ODS('GetLastQuarter       Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107

GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.8 GetLastWeek

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start date and end date of the last/previous week based on the pDate base date. This is from Monday to Sunday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLastWeek(D,FD,TD)
 ITD.ODS('GetLastWeek          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95

GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.9 GetLastWorkWeek

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start date and end date of the last/previous work week based on the pDate base date. This is from Monday to Friday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLastWorkWeek(D,FD,TD)
 ITD.ODS('GetLastWorkWeek      Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetNextWorkWeek 106
GetThisMonth 111

GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.10 GetLastYear

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the dates for January 1 and December 31 in the last/previous year based on the pDate base date.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetLastYear(D,FD,TD)
 ITD.ODS('GetLastYear          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104

3.5.3.11 GetMonthFromDate

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start and end dates for the current month specified by the pDate base date but only from the current date specified by pDate. pFromDate will be equal to pDate.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetMonthFromDate(D,FD,TD)
 ITD.ODS('GetMonthFromDate     Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.12 GetMonthName

Prototype:	**(Byte pMonth, Byte pLongName=True),String**
pMonth	Month value to get the name for, must be in the range of 1 to 12.
pLongName	Determine if it should return the long/full month name. This parameter is true by default.
Returns	The name of the month as set by InitMonthNames or overridden by SetMonthName.

This method is used to retrieve the month name, such as "January" based on a month number, 1 for January, 2 for February and so on. The Constructor 122 calls the InitMonthNames 120 method which sets the English month names, both long and short (3 character abbreviated month names, Jan, Feb, etc.)

Example:

```
ITD  ITDateClass
 Code
 ITD.SetMonthName(1,'Janúar','Jan')  !! Set Icelandic name for January
 Message('January is now: ' & ITD.GetMonthName(1))
```

See also:
InitMonthNames 120
SetMonthName 121
Construct 122

3.5.3.13 GetMonthToDate

Prototype:	**(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**
pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start and end dates for a period that started at the beginning of the current month specified by the pDate base date. pToDate will be equal to pDate and pFromDate will be equal to the 1st day of the month.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetMonthToDate(D,FD,TD)
 ITD.ODS('GetMonthToDate       Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115

GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.14 GetNextMonth

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start date and end date of the next month based on the pDate base date. Leap years do not affect this method and neither do changes in years, i.e. if pDate is in December GetLastMonth will return the first and last dates for January 1 and 31 the year after.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()

 Ds = ITD.GetNextMonth(D,FD,TD)
 ITD.ODS('GetNextMonth         Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94

GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.15 GetNextQuarter

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.

Returns Number of days in the period

This method returns the first and last date in the next quarter based on the pDate base date. If the current quarter is the last quarter, this method will return the dates for January 1 and March 31, the year after.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetNextQuarter(D,FD,TD)
 ITD.ODS('GetNextQuarter       Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114

3.5.3.16 GetNextWeek

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start date and end date of the next week based on the pDate base date. This is from Monday to Sunday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetNextWeek(D,FD,TD)
 ITD.ODS('GetNextWeek          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

3.5.3.17 GetNextWeekDay

Prototype: **(Long pBaseDate, Byte pWeekDay, Byte pDayIfSame=True),LONG**

pBaseDate Base date to calculate from

pWeekDay The day of the week to find, IT_Sunday - IT_Saturday

pDayIfSame What to do if the base date is a pWeekDay, if True, then it returns the pBaseDate, if False it returns pBaseDate plus 7, i.e. same day of week next week.

Returns The calculated weekday

This method is used to find the next weekday, for example to find what day next Tuesday is calculated from the pBaseDate date. If the pBaseDate happens to be the weekday that we are looking for then pDayIfSame determines if the method returns the pBaseDate or if it returns the pBaseDate plus 7 for next weeks date. Day name constants are declared in the ITEquates.inc file to make day selections easy, IT_Sunday, IT_Monday, IT_Tuesday, IT_Wednesday, IT_Thursday, IT_Friday, IT_Saturday.

Example:

```
ITD  ITDateClass
D    Date
 Code
 D = Date(7,24,2010)  !! Saturday
```

```
 ITD.ODS('Base date is:           ' & Format(D,@d18))
 ITD.ODS('Next weeks Saturday is: ' &
Format(ITD.GetNextWeekDay(D,IT_SaturDay,False),@d18))
 ITD.ODS('Next weeks Saturday is: ' &
Format(ITD.GetNextWeekDay(D,IT_SaturDay),@d18))

! Results:
Base date is:           July 24, 2010
Next weeks Saturday is: July 31, 2010  !! Get next week if dates match.
Next weeks Saturday is: July 24, 2010  !! Get base date if dates match.
```

See also:
GetPreviousWeekDay 108

3.5.3.18 GetNextWorkWeek

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start date and end date of the next work week based on the pDate base date. This is from Monday to Friday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetNextWorkWeek(D,FD,TD)
 ITD.ODS('GetNextWorkWeek      Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102

GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.19 GetNextYear

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the dates for January 1 and December 31 in the next year based on the pDate base date.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetNextYear(D,FD,TD)
 ITD.ODS('GetNextYear          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114

3.5.3.20 GetPreviousWeekDay

Prototype: **(Long pBaseDate, Byte pWeekDay, Byte pDayIfSame=True),LONG**

pBaseDate Base date to calculate from

pWeekDay The day of the week to find, IT_Sunday - IT_Saturday

pDayIfSame What to do if the base date is a pWeekDay, if True, then it returns the pBaseDate, if False it returns pBaseDate plus 7, i.e. same day of week next week.

Returns The calculated weekday

This method is used to find the previous weekday, for example to find what day last Tuesday is calculated from the pBaseDate date. If the pBaseDate happens to be the weekday that we are looking for then pDayIfSame determines if the method returns the pBaseDate or if it returns the pBaseDate plus 7 for last weeks date. Day name constants are declared in the ITEquates.inc file to make day selections easy, IT_Sunday, IT_Monday, IT_Tuesday, IT_Wednesday, IT_Thursday, IT_Friday, IT_Saturday.

Example:

```
ITD  ITDateClass
D    Date
 Code
 D = Date(7,24,2010)  !! Saturday
 ITD.ODS('Base date is:           ' & Format(D,@d18))
 ITD.ODS('Next weeks Saturday is: ' &
Format(ITD.GetNextWeekDay(D,IT_SaturDay,False),@d18))
 ITD.ODS('Next weeks Saturday is: ' &
Format(ITD.GetNextWeekDay(D,IT_SaturDay),@d18))
```

```
! Results:
Base date is:           July 24, 2010
Last weeks Saturday is: July 17, 2010  !! Get last week if dates match.
Last weeks Saturday is: July 24, 2010  !! Get base date if dates match.
```

See also:
GetPreviousWeekDay 108

3.5.3.21 GetQuarterFromDate

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start and end dates for the current quarter specified by the pDate base date but only from the current date specified by pDate. pFromDate will be equal to pDate.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetQuarterFromDate(D,FD,TD)
 ITD.ODS('GetQuarterFromDate   Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95

GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetYearFromDate 118

3.5.3.22 GetQuarterToDate

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start and end dates for the current quarter specified by the pDate base date but only up to the current date specified by pDate. pToDate will be equal to pDate.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetQuarterToDate(D,FD,TD)
 ITD.ODS('GetQuarterToDate     Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111

GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.23 GetThisMonth

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start date and end date of the current month based on the pDate base date. Leap years do not affect this method.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetThisMonth(D,FD,TD)
 ITD.ODS('GetThisMonth         Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104

3.5.3.24 GetThisQuarter

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the first and last date in the current quarter based on the pDate base date.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetThisQuarter(D,FD,TD)
 ITD.ODS('GetThisQuarter       Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

3.5.3.25 GetThisWeek

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start date and end date of the current week based on the pDate base date. This is from Monday to Sunday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetThisWeek(D,FD,TD)
 ITD.ODS('GetThisWeek          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
```

```
Ds)
```

See also:
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.26 GetThisWorkWeek

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start date and end date of the current work week based on the pDate base date. This is from Monday to Friday.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
```

```
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetThisWorkWeek(D,FD,TD)
 ITD.ODS('GetThisWorkWeek      Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.27 GetThisYear

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the dates for January 1 and December 31 in the current year based on the pDate base date.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetThisYear(D,FD,TD)
 ITD.ODS('GetThisYear          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.28 GetWeekFirstDay

Prototype: **(Date pStartDate, Long pWeek),Long**

pStartDate Base date to calculate from.

pWeek Week to move to forward or backward. This can be negative, zero or positive number. Negative number moves backward, 0 returns the first day of the current week and positive number moved forward.

Returns The first day of the specified week

This method will return the first day of a specified week, which can be in the past, present or future compared to the pStartDate. For example to get the first day of last week, you would use -1 as the pWeek parameter. To get next week's first day, you would use 1.

Example:

```
ITD  ITDateClass
FD   Date
D    Date
 Code
 D = Date(7,28,2010)
 ITD.SetWeekStartDay(IT_Sunday)  !! Set the first day of the week to Sunday

 FD = ITD.GetWeekFirstDay(D,-1)
 ITD.ODS('First day -1: ' & Format(FD,@d18))  !! July 18, 2010

 FD = ITD.GetWeekFirstDay(D,0)
 ITD.ODS('First day 0: ' & Format(FD,@d18))   !! July 25, 2010

 FD = ITD.GetWeekFirstDay(D,1)
 ITD.ODS('First day 1: ' & Format(FD,@d18))   !! August 01, 2010
```

See also:
SetWeekStartDay 122
GetWeekStartDay 117

3.5.3.29 GetWeekNumber — Date Class - Methods

Prototype: **(Date pDate),Byte**

pDate Standard Clarion date to get the week number for.

Returns The weeknumber for the date given in pDate. This value corresponds to ISO 8601 standard week calculations.

This method returns the week number for a given date according to ISO 8601 standard. It dictates that a week begins on a Monday and the first week of the year is the week containing the first Thursday after January 1st or equivalently the week that includes January 4th.

Example:

```
ITD  ITDateClass
 Code
 Message('The week number for ' & Format(Today(),@d18) & ' is ' &
ITD.GetWeekNumber(Today()))
```

See also:
http://en.wikipedia.org/wiki/ISO_8601#Week_dates

3.5.3.30 GetWeekStartDay — Date Class - Methods

Prototype: **(),Byte**

Returns The week start day in the range of IT_Sunday to IT_Saturday (0 - 6)

This method is used to query the WeekStartDay 88 property.

Example:

```
ITD  ITDateClass
 Code
 ITD.ODS('The week starts on ' & ITD.GetWeekStartDate())
```

See also:
SetWeekStartDay 122
WeekStartDay 88

3.5.3.31 GetYearFromDate

Date Class - Methods

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate Base date to calculate from.
pFromDate Start date of the period.
pToDate End date of the period.

Returns Number of days in the period

This method returns the start and end dates for the current year specified by the pDate base date but only from the current date specified by pDate. pFromDate will be equal to pDate.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetYearFromDate(D,FD,TD)
 ITD.ODS('GetYearFromDate      Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetThisWeek 113
GetLastWeek 96
GetNextWeek 104
GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94

GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetYearToDate 119
GetMonthFromDate 99
GetQuarterFromDate 109

3.5.3.32 GetYearToDate

Prototype: **(Date pDate, *ANY pFromDate, *ANY pToDate),LONG,PROC**

pDate	Base date to calculate from.
pFromDate	Start date of the period.
pToDate	End date of the period.
Returns	Number of days in the period

This method returns the start and end dates for the current year specified by the pDate base date but only up to the current date specified by pDate. pToDate will be equal to pDate.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 Ds = ITD.GetYearToDate(D,FD,TD)
 ITD.ODS('GetYearToDate        Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:

GetThisWeek 113
GetLastWeek 96
GetNextWeek 104

GetThisWorkWeek 114
GetLastWorkWeek 97
GetNextWorkWeek 106
GetThisMonth 111
GetLastMonth 94
GetNextMonth 102
GetThisQuarter 112
GetLastQuarter 95
GetNextQuarter 103
GetThisYear 115
GetLastYear 98
GetNextYear 107
GetLast12Months 92
GetMonthToDate 101
GetQuarterToDate 110
GetMonthFromDate 99
GetQuarterFromDate 109
GetYearFromDate 118

3.5.3.33 InitDayNames

Prototype: **(none)**

This method is called from the constructor 122 and doesn't really need to be called. It sets up the English day names, i.e. Monday, Tuesday, etc. The day names can be modified at any time by using the SetDayName 121 method. The day names can then be retrieved by using the GetDayName.

See also:
SetDayName 121
GetDayName 92

3.5.3.34 InitMonthNames

Prototype: **(none)**

This method is called from the constructor 122 and doesn't really need to be called. It sets up the English month names, i.e. January, February etc. The month names can be modified at any time by using the SetMonthName 121 method.

See also:
SetMonthName 121

3.5.3.35 SetDayName

Date Class - Methods

Prototype: **(Byte pDay, String pDayName, <String pShortName>)**

pDay	The day of the week to change the name for. IT_Sunday - IT_Saturday.
pDayName	The full day name, i.e. long name for example 'Monday'
[pShortName]	Omittable short, abbreviated name for the day, for example 'Mon'

This method is used to set the day names for example to a different language. Day name constants are declared in the ITEquates.inc file to make day selections easy, IT_Sunday, IT_Monday, IT_Tuesday, IT_Wednesday, IT_Thursday, IT_Friday, IT_Saturday.

Example:

```
ITD  ITDateClass
 Code
 ITD.SetDayName(IT_Saturday,'Laugardagur', 'Lau')  !! Set to Icelandic for Saturday
 Message('Saturday is now: ' & ITD.GetDayName(IT_Saturday))
```

See also:

GetDayName 92

InitDayName 120

GetNextWeekDay 105

GetPreviousWeekDay 108

3.5.3.36 SetMonthName

Date Class - Methods

Prototype: **(Byte pMonth, String pMonthName, <String pShortName>)**

pMonth	Value 1 - 12 indicating the month to set the name for.
pMonthName	Long/full name for the month, for example 'January'
[pShortName]	Omittable short, abbreviated name for the month, for example 'Jan'

This method is used to set the month names for example to a different language. The month name will then be returned when calling the GetMonthName 100 method.

Example:

```
ITD  ITDateClass
 Code
 ITD.SetMonthName(1,'Janúar','Jan')  !! Set Icelandic name for January
 Message('January is now: ' & ITD.GetMonthName(1))
```

See also:

GetMonthName 100

InitMonthNames 120

3.5.3.37 SetWeekStartDay

Date Class - Methods

Prototype: **(Byte pStartDay)**

pStartDay Day value in the range of IT_Sunday to IT_Saturday (0 - 6) indicating what day of the week is considered to the week's start date.

This method is used to set the week start day to any day in the IT_Sunday - IT_Saturday range, indicating the day that the week starts on. This method is called from the Construct 122 method to set the WeekStartDay 88 property to IT_Monday according to ISO 8601.

Example:

```
ITD  ITDateClass
FD   Date
TD   Date
D    Date
Ds   Long
 Code
 D = Today()
 ITD.SetWeekStartDate(IT_Sunday)
 Ds = ITD.GetThisWeek(D,FD,TD)
 ITD.ODS('GetThisWeek          Date: ' & Format(D,@d18) & ', From: ' &
Format(Format(FD,@d18),@s30) & 'To: ' & Format(Format(TD,@d18),@s30) & 'Days: ' &
Ds)
```

See also:
GetWeekStartDay 117
WeekStartDay 88

3.5.3.38 Construct

Date Class - Methods

Prototype: **(none)**

The Constructor 122 in the Date Class sets up two queues and assigns 3 properties. It creates Months 88 and Days 87 queues and calls InitMonthNames 120 and InitDayNames 120 to initialize the month and day names. It also fills the Q1 88, Q2 88 and DE 87 dimensioned properties that are used in the Quarter calculation methods such as GetThisQuarter 112, GetLastQuarter 95, GetNextQuarter 103, GetQuarterToDate 110 and GetQuarterFromDate 109 methods.

See also:
Destruct 122

3.5.3.39 Destruct

Date Class - Methods

Prototype: **(none)**

The Destructor in the Date Class Frees and disposed of the 2 queues that the constructor creates, Months 88 and Days 87.

See also:
Construct 122

3.6 Debug Class

3.6.1 Overview Debug Class

```
ITDebugClass
CLASS(ITUtilityClass),TYPE,Module('ITDebugClass.clw'),Link('ITDebugClass',_ITUtilLi
nkMode_),DLL(_ITUtilDllMode_)

UseFile                    Byte
UseODS                     Byte
OutputFile                 CString(1025)
Output                     &ITDebugQueue
ODS                        Procedure(String pS),VIRTUAL
Init                       Procedure(Byte pUseFile=False, Byte pAlsoODS=True,
<String pFileName>)
WriteToFile                Procedure(Byte pAppend=True)
Construct                  Procedure
Destruct                   Procedure
                         END
```

3.6.2 Properties Debug Class

Enter topic text here.

3.6.3 Methods Debug Class

Enter topic text here.

3.7 Directory Class

3.7.1 Overview Directory Class

```
ITDirectoryClass
CLASS(ITUtilityClass),TYPE,Module('ITDirectoryClass.clw'),Link('ITDirectoryClass',_
ITUtilLinkMode_),DLL(_ITUtilDllMode_)

FileQueue                &ITFileQueue
ReadDir                  CString(2049) ! Path part of the pPath passed to
ReadDirectory
ReadFiles                CString(129)  ! Filename + Extension part of pPath passed
to ReadDirectory
NumberOfFiles            Long
ReadDirectory            Procedure(String pPath, Long pAttrib=ff_:normal, Byte
pFree=True),Long,Proc
LowercaseExtension       Procedure
DumpInQueue              Procedure(Queue pQ, <*? pFName>,<*? pFSize>,<*?
pFDate>,<*? pFTime>,<*? pFAttrib>,<*? pFullName>)
Construct                Procedure
Destruct                 Procedure
                        END
```

3.7.2 Properties Directory Class

Enter topic text here.

3.7.3 Methods Directory Class

Enter topic text here.

3.8 EXIF Class

3.8.1 Overview EXIF Class

This class uses an activeX from http://www.watermarker.com/ Unfortunately this Active-X is slow and we are going to rewrite this class to use the FreeImage library. If you have the Active-X you can use this class to extract EXIF information from image files.

```
ITExifClass
CLASS(ITStringClass),TYPE,Module('ITExifClass.clw'),Link('ITExifClass',_ITUtilLinkM
ode_),DLL(_ITUtilDllMode_)

! This uses the aisExif.dll from http://www.watermarker.com/
Exif                    &ITExifQ
ExifList                &String
LoadParameters          Procedure,VIRTUAL
ReadExifInfo            Procedure(String pImageName)
GetParam                Procedure(String pParam),String
SetParam                Procedure(String pImageName, String pParam, String
pValue)
DumpInQueue             Procedure(Queue pQ, *? pParam, *? pValue, Byte
pFilledOnly=True, |
                                  Byte pUseDescription=True, Byte
pConvertDates=True),VIRTUAL
AddQueueEntry           Procedure(String pParameter, Queue pQ, *? pParam, *?
pValue, |
                                  Byte pUseDescription=True, Byte
pConvertDates=True),VIRTUAL
GetDateTimeValue        Procedure(String pDateTime),String,VIRTUAL
GetDateValue            Procedure(String pDateTime),Long
GetTimeValue            Procedure(String pDateTime),Long
Construct               Procedure
Destruct                Procedure
                      END
```

3.8.2 Properties EXIF Class

Enter topic text here.

3.8.3 Methods EXIF Class

Enter topic text here.

3.9 Export Class

3.9.1 Overview Export Class

```
ITExportClass
Class(ITUtilityClass),TYPE,Module('ITExportClass.clw'),Link('ITExportClass',_ITUtil
LinkMode_),DLL(_ITUtilDllMode_)

FileRecord                &GROUP
ExportedFile              &File
ExportedView              &View
ExportFile                String(IT_MAX_Path)
AllFields                 &FieldQueueType
ExpFields                 &FieldQueueType
NumberOfFields            Long
Initialized               Byte
ExportReady               Byte
RecordsExported           Long
QuoteCharacter            String(1),PRIVATE
DelimiterCharacter        String(1),PRIVATE

Init                      Procedure(FILE pFile)
Init                      Procedure(VIEW pView)
Init                      Procedure,PRIVATE
GetNumberOfFields         Procedure(FILE pFile),LONG
LoadFileFields            Procedure(FILE pFile),LONG,PROC
LoadViewFields            Procedure(),LONG,PROC
AddExportField            Procedure(STRING pFieldName, <STRING pHeaderName>, <BYTE
pQuote>, <BYTE pIsVar>, <BYTE pVarIsNum>),BYTE,PROC
SetQuoteCharacter         Procedure(STRING pQuoteChar)
SetDelimiter              Procedure(STRING pDelimiter)
GetQuoteCharacter         Procedure(),STRING
GetDelimiter              Procedure(),STRING
StartExport               Procedure(STRING pExportFile, Byte pWriteHeaders=True,
Byte pQuoteHeaders=True)
WriteHeaders              Procedure(BYTE pQuote=True, <STRING pDelimiter>, <STRING
pQuoteWith>),BYTE,PROC
                                    ! Returns true if successful, false if it
failed.
ExportRecord              Procedure(),BYTE,PROC  ! Returns true if successful,
false if it failed.
WriteLine                 Procedure(String pLine),BYTE,PROC ! Writes an
unformatted line to the file
EndExport                 Procedure(),LONG,PROC  ! Returns number of records
exported successfully.
ParseHeaderNameFromField  Procedure(STRING pFieldName),STRING
Kill                      Procedure()
                        End
```

3.10 File Class

3.10.1 Overview File Class

The File Class contains methods that give you information about drives. It should probably have been named Drive Class, but in fact it started out with more methods related to files, which have been moved to the Core Class and other classes for easier access.

```
ITFileClass
Class(ITNetworkClass),TYPE,Module('ITFileClass.clw'),Link('ITFileClass',_ITUtilLink
Mode_),DLL(_ITUtilDllMode_)
LocalDrives 128                &IT_LocalDrives

EnumLocalDrives 129            Procedure(),Long  ! Returns the number of enumerated
drives
GetDriveType 129               Procedure(String pDrive),Long
GetDriveTypeString 130         Procedure(String pDrive),String
GetDriveSerialNumber 131       Procedure(String pDrive),String
GetVolumeInfo 132              Procedure(String pDrive, Long pInfoToGet),String
IsLocalDrive 132               Procedure(String pPath),Byte  ! Takes drive, path or
path+file and returns true/false

Construct 133                  Procedure
Destruct  133                  Procedure
                          End
```

3.10.2 Properties File Class

There is only one property in the File Class, which is used to keep track of enumerated local drives.

```
LocalDrives 128                &IT_LocalDrives
```

3.10.2.1 LocalDrives File Class - Properties

A queue of type IT_LocalDrives:

```
IT_LocalDrives           QUEUE,TYPE
DriveLetter                String(1)
RootPath                   String(255)
DriveType                  Long
DriveVolumeInfo            GROUP(IT_DriveVolumeInfo)
                           END
                         END
```

3.10.3 Methods File Class

There are currently 8 methods in the File Class:

```
EnumLocalDrives 129            Procedure(),Long  ! Returns the number of enumerated
drives
GetDriveType 129               Procedure(String pDrive),Long
GetDriveTypeString 130         Procedure(String pDrive),String
GetDriveSerialNumber 131       Procedure(String pDrive),String
GetVolumeInfo 132              Procedure(String pDrive, Long pInfoToGet),String
IsLocalDrive 132               Procedure(String pPath),Byte  ! Takes drive, path or
```

```
path+file and returns true/false

Construct 133                   Procedure
Destruct  133                   Procedure
```

3.10.3.1 EnumLocalDrives

File Class - Methods

Prototype: **(),LONG,PROC**

Returns Returns number of logical drives.

This method enumerates all logical drives that are local to the computer. This includes diskette drives, hard drives, CD/DVD drives, removable drives and remote mapped drives.

Example:

```
ITF  ITFileClass
I    Long
 ITF.EnumLocalDrives()
 Loop I = 1 To Records(ITF.LocalDrives)
   Get(ITF.LocalDrives,I)
   ITF.ODS('Drive: ' & Clip(ITF.LocalDrives.RootPath) &|
           ', Type = ' & ITF.LocalDrives.DriveType    &|
           ': ' & ITF.GetDriveTypeString(ITF.LocalDrives.RootPath))
 End
```

See also:
GetDriveType 129
GetDriveTypeString 130

3.10.3.2 GetDriveType

File Class - Methods

Prototype: **(String pDrive),Long**

pDrive Drive letter to get the drive type for. Normally passed as C:\

Returns Returns the drive type

This method returns an integer drive type for the specified drive. The value can be one of the following:

Value	Meaning
IT_DRIVE_UNKNOWN	The drive type cannot be determined.
IT_DRIVE_NO_ROOT_DIR	The root path is invalid, for example, no volume is mounted at the path.
IT_DRIVE_REMOVABLE	The drive is a type that has removable media, for example, a floppy drive or removable hard disk.

IT_DRIVE_FIXED	The drive is a type that cannot be removed, for example, a fixed hard drive.
IT_DRIVE_REMOTE	The drive is a remote (network) drive.
IT_DRIVE_CDROM	The drive is a CD-ROM drive.
IT_DRIVE_RAMDISK	The drive is a RAM disk.

To get a text description of each type use the GetDriveTypeString 130 method.

Example:

```
ITF  ITFileClass
I    Long
 ITF.EnumLocalDrives()
 Loop I = 1 To Records(ITF.LocalDrives)
   Get(ITF.LocalDrives,I)
   ITF.ODS('Drive: ' & Clip(ITF.LocalDrives.RootPath) &|
           ', Type = ' & ITF.LocalDrives.DriveType    &|
           ': ' & ITF.GetDriveTypeString(ITF.LocalDrives.RootPath))
 End
```

See also:
EnumLocalDrives 129
GetDriveTypeString 130

3.10.3.3 GetDriveTypeString

Prototype: **(String pDrive),String**

pDrive Drive letter to get the drive type string for. Normally passed as C:\

Returns Return string with the drive type

This method takes the drive letter string and returns a string that van be one of the following values:

Value	Meaning
Unknown drive type	The drive type cannot be determined.
No root directory	The root path is invalid, for example, no volume is mounted at the path.
Removable	The drive is a type that has removable media, for example, a floppy drive or removable hard disk.
Fixed	The drive is a type that cannot be removed, for example, a fixed hard drive.

Remote	The drive is a remote (network) drive.
CD-ROM	The drive is a CD-ROM drive.
RAMDISK	The drive is a RAM disk.

Example:

```
ITF  ITFileClass
I    Long
 ITF.EnumLocalDrives()
 Loop I = 1 To Records(ITF.LocalDrives)
   Get(ITF.LocalDrives,I)
   ITF.ODS('Drive: ' & Clip(ITF.LocalDrives.RootPath) &|
           ', Type = ' & ITF.LocalDrives.DriveType    &|
           ': ' & ITF.GetDriveTypeString(ITF.LocalDrives.RootPath))
 End
```

See also:
GetDriveType 129
EnumLocalDrive 129

3.10.3.4 GetDriveSerialNumber — File Class - Methods

Prototype: **(String pDrive),String**

pDrive Drive letter to get the drive type string for. Normally passed as C:\ or \\servername\share\

Returns Returns drive serial number formatted as xxxx-xxxx.

This method calls the GetVolumeInfo 132 method to extract the volume serial number from a drive. Note that this is NOT hardware serial number, just the volume serial number that is created when the partition is formatted.

Example:

```
ITF  ITFileClass
 Code
 ITF.ODS('Serial number = ' & ITF.GetDriveSerialNumber('c:\'))
```

This will return the serial number for the C: drive.

See also:
GetVolumeInfo 132

3.10.3.5 GetVolumeInfo

File Class - Methods

Prototype: **(String pDrive, Long pInfoToGet),String**

pDrive Drive letter to get the drive type string for. Normally passed as C:\ or \\servername\share\

pInfoToGet Information equate determining which volume information to get. See below.

Returns Returns the requested information.

This method can be called with several different options to return different information.

IT_VolInfo:VolName EQUATE
IT_VolInfo:VolSerial EQUATE
IT_VolInfo:VolCompLen EQUATE
IT_VolInfo:VolSysFlags EQUATE
IT_VolInfo:VolSysName EQUATE

Value	Meaning
IT_VolInfo:VolName	Returns the volume name (string)
IT_VolInfo:VolSerial	Returns the volume serial number (string) - see GetDriveSerialNumber 131
IT_VolInfo:VolCompLen	Returns the component length (integer) such as filename length.
IT_VolInfo:VolSysFlags	Returns the volume system flags. See http://msdn.microsoft.com/en-us/library/aa364993(VS.85).aspx for more information (integer)
IT_VolInfo:VolSysName	Returns the volume system name, such as FAT32 or NTFS (string).

See also:
GetDriveSerialNumber 131

3.10.3.6 IsLocalDrive

File Class - Methods

Prototype: **(String pPath),Byte**

pPath Path to the drive root. This can be a full file path.

Returns True or false depending on if the drive is a local drive or not

This method checks if a drive is a local drive or not. A local drive is a fixed drive, removable drive, CD/DVD and RAM disk.

Example:

```
ITF  ITFileClass
 Code
 If ITF.IsLocalDrive('c:\')
   Message('C:\ is local drive')
 Else
   Message('C:\ is NOT local drive')
 End
```

See also:
GetFilePart 65
Is98NetCompatible 181
GetDriveType 129

3.10.3.7 Construct — File Class - Methods

Prototype: **None**

The construct creates an instance of the LocalDrives 128 queue.

See also:
Destruct 133

3.10.3.8 Destruct — File Class - Methods

Prototype: **(none)**

The Destruct destroys the instance of the LocalDrives 128 queue.

See also:
Construct 133

3.11 File Search Class

3.11.1 Overview File Search Class

The file search class is designed to search for files and folders.

```
ITFileSearchClass
Class(ITFileClass),TYPE,Module('ITFileSearchClass.clw'),Link('ITFileSearchClass',_I
TUtilLinkMode_),DLL(_ITUtilDllMode_)
Directories 136                 &ITDirQueue 135
Files 137                       &ITFileQueueLS 135   !!ITFileQueue
WildCards 139                   &ITWildcards 135
StartDirectory 138              CString(2049)
TotalDirectories 138            Long
TotalFiles 138                  Long
FindHandle 137                  IT_HANDLE
FileSort 137                    Byte
FileFilter 137                  CString(256)
TotalFileSize 138               DECIMAL(15,0)  !! AB 2010-01-06:  Enable size > 2GB
TotalFileStringSize 138         Long
FileAttributes 519              Long

CountFilesInDirectories 139     Procedure(String pFF,<String pDirectory>),Long
GetFileSort 140                 Procedure(), Byte
GetLevel 140                    Procedure(String pDir),Byte
GetWildcardList 141             Procedure(String pWildcards),Long,Proc
Init 214                        Procedure(Long pFileAttributes)
ReadDirectories 142             Procedure(String pDir, Byte pShowWindow=False),Private
ResetFileCounters 142           Procedure(<String pDirectory>)
ScanDirectories 143             Procedure(<String pSD>, Byte pShowWindow=False, Byte
pRecurse=True),Long,Proc
ScanFiles 143                   Procedure(<String pDir>,<String pWC>, Long
pAttrib=FF_:NORMAL, BYTE pCountOnly=False),Long,Proc
SetFileFilter 145               Procedure(String pFF)
SetFileSort 145                 Procedure(Byte pSort)
SetNoFileSort 146               Procedure
SetStartDir 146                 Procedure(String pSD)
Construct 48                    Procedure
Destruct 48                     Procedure
                              End
```

3.11.2 Data Types File Search Class

The File Search class uses 3 data types:

```
ITFileQueueLS 135                QUEUE,PRE(ITDLS),TYPE
Name                           STRING(FILE:MaxFileName)
ShortName                      STRING(13)
Date                           LONG
Time                           LONG
Size                           DECIMAL(15,0)
Attrib                         BYTE
FullName                       CString(2049)
DirRef                         Long
                             END
```

```
ITDirQueue 135                QUEUE,TYPE
FullName                      CString(2049)
DirectoryName                 CString(2049)
FilesLoaded                   Byte
SubLevel                      Byte
FileCounter                   Long
                            END

ITWildcards 135               QUEUE,TYPE
WC                            CString(20)
                            END
```

3.11.2.1 ITDirQueue

File Search Class - Data Types

This queue type is declared as Directories and is used in the ScanDirectories and ReadDirectories methods.

```
ITDirQueue                  QUEUE,TYPE
FullName                      CString(2049)
DirectoryName                 CString(2049)
FilesLoaded                   Byte
SubLevel                      Byte
FileCounter                   Long
                            END
```

The Fullname field contains the full path and name of the folder, such as "C:\Program Files\Icetips" where as the DirectoryName contains just the deepest level name, i.e. "Icetips" The SubLevel field contains the number of directory levels "down" the current directory is. In the case of "C:\Program Files\Icetips", "Icetips" would be at level 2. This can be determined on any path string by using the GetLevel 140 method, but please note that it is **not UNC compatible**.

3.11.2.2 ITFileQueueLS

File Search Class - Data Types

Same as ITFileQueue except Size is re-declared. Cannot be used with DIRECTORY

```
ITFileQueueLS               QUEUE,PRE(ITDLS),TYPE
Name                          STRING(FILE:MaxFileName)
ShortName                     STRING(13)
Date                          LONG
Time                          LONG
Size                          DECIMAL(15,0)
Attrib                        BYTE
FullName                      CString(2049)
DirRef                        Long
                            END
```

The Files property is an instance of ITFileQueueLS and it is used in the ScanFiles method to store file information. The Size field in the queue is assigned with GetFileSize 66 which support file sizes that are larger than 2GB which is the limit with DIRECTORY.

See also:
GetFileSize 66

3.11.2.3 ITWildcards

File Search Class - Data Types

This queue is used for the Wildcards 139 property and is filled with the GetWildcardList 141 which takes a semicolon separated string and splits it up into multiple wildcards, such as *.exe;*.dll would become two records with *.exe and *.dll respectively. It is only used in the CountFilesInDirectories 139 and the

GetWildcardList 141 methods.

```
ITWildcards                 QUEUE,TYPE
WC                            CString(20)
                            END
```

3.11.3 Properties — File Search Class

The File Search class has 12 properties.

```
Directories 136               &ITDirQueue
Files 137                     &ITFileQueueLS  !!ITFileQueue
WildCards 139                 &ITWildcards
StartDirectory 138            CString(2049)
TotalDirectories 138          Long
TotalFiles 138                Long
FindHandle 137                IT_HANDLE
FileSort 137                  Byte
FileFilter 137                CString(256)
TotalFileSize 138             DECIMAL(15,0)  !! AB 2010-01-06:  Enable size > 2GB
TotalFileStringSize 138       Long
FileAttributes 136            Long
```

3.11.3.1 Directories — File Search Class - Properties

The Directories property is used in the ScanDirectories 143, ReadDirectories 142, CountFilesInDirectories 139, ResetFileCounters 142 and ScanFiles 143 methods and holds information about the directories that are being scanned in by ScanDirectories 143 or ReadDirectories 142.

It is declared as:

```
Directories                 &ITDirQueue
```

and instantiated as:

```
SELF.Directories &= NEW ITDirQueue 135
```

in the Construct 146 method.

3.11.3.2 FileAttributes — File Search Class - Properties

The FileAttributes property is used in the Init 141, CountFilesInDirectories 139, ScanFiles 143 and Construct 146 methods and is used to set the default file attributes that are scanned for in the ScanFiles 143. I.e. it is used as the Attribute parameter in the Directory statement that reads the files in the ScanFiles 143 method. In the Construct 146 method the FileAttributes is set to FF_:NORMAL. It can be set by the Init method, that only sets the FileAttributes.

It is declared as:

```
FileAttributes              Long
```

3.11.3.3 FileFilter

The FileFilter property is used in the SetFileFilter 145 and ScanFiles 143 methods to set the wildcards for the files scanned by ScanFiles 143. By default it is *.* You can change it by calling SetFileFilter 145 before calls are made to ScanFiles 143.

It is declared as:

```
FileFilter                  CString(256)
```

3.11.3.4 Files

The Files property is used in the ScanFile 143 method and holds information about the files that have been read in Scanfiles 143. It is based on the ITFileQueueLS 135 datatype.

It is declared as:

```
Files                       &ITFileQueueLS
```

and instantiated as:

```
SELF.Files       &= NEW ITFileQueueLS
```

in the Construct 146 method.

3.11.3.5 FileSort

The FileSort property is used in the SetFileSort 145, GetFileSort 140 and SetNoFileSort 146, ScanFiles 143 methods and is used to specify how the files read by ScanFiles 143 are sorted in the Files 137 queue property. The possible sort orders are itemized in IT_Filesort in the ITEquates.inc file:

```
IT_FileSort              ITEMIZE(1),PRE(IT_FileSort)
Name                       EQUATE
ShortName                  EQUATE
Date                       EQUATE
Time                       EQUATE
Size                       EQUATE
Attrib                     EQUATE
                         END
```

The FileSort property is declared as:

```
FileSort                    Byte
```

3.11.3.6 FindHandle

The FindHandle property is **not** used.

It is declared as:

```
FindHandle                  IT_HANDLE
```

3.11.3.7 StartDirectory

The StartDirectory property is used in the ScanDirectories 143 and SetStartDir 146 methods and is used to keep the start directory for the ScanDirectories 143 method. If a path is passed to the ScanDirectories 143 method it is assigned to the StartDirectory property. It is used in the initial call to the ReadDirectories 142. StartDirectory is a CString datatype.

It is declared as:

```
StartDirectory              CString(2049)
```

3.11.3.8 TotalDirectories

The TotalDirectories property is used in the ScanDirectories 143 and ReadDirectories 142 methods and is used to store the total number of directories read into the Directories 136 queue.

It is declared as:

```
TotalDirectories            Long
```

3.11.3.9 TotalFiles

The TotalFiles property is used in the ScanDirectories 143 and ReadDirectories 142 methods and is used to store the number of files read.

It is declared as:

```
TotalFiles                  Long
```

3.11.3.10 TotalFileSize

The TotalFileSize property is used in the ScanFiles 143 method to keep the total file size in the directory or directories being scanned. If ScanFiles 143 is called from the CounFilesInDirectories 139, i.e. if ScanFiles 143 is called with the pCountOnly=True, then TotalFileSize is not used and will be 0 (zero).

It is declared as:

```
TotalFileSize               DECIMAL(15,0)
```

3.11.3.11 TotalFileStringSize

The TotalFileStringSize property is used in the ScanFiles 143 method and is used to keep a track of how long string size would be needed to keep all the filenames, with paths, in a single string. This can be useful if the purpose of the file search is to create a string or a file with the search results. Then you will need a string that is TotalFileStringSize long. Note that the TotalFileStringSize does not include any end of line characters.

It is declared as:

```
TotalFileStringSize         Long
```

3.11.3.12 WildCards

File Search Class - Properties

The WildCards property is used in the CountFilesInDirectories 139 and GetWildcardList 141 methods and holds information about the wildcards that will be used in the CountFilesInDirectories 139 method. It is based on the ITWildcards 135 datatype.

It is declared as:

```
WildCards                   &ITWildcards 135
```

and instantiated as:

```
SELF.WildCards   &= NEW ITWildcards 135
```

in the Construct 146 method.

3.11.4 Methods — File Search Class

The File Search class has 15 methods including constructor and destructor.

```
CountFilesInDirectories 139    Procedure(String pFF,<String pDirectory>),Long
GetFileSort 140                Procedure(), Byte
GetLevel 140                   Procedure(String pDir),Byte
GetWildcardList 141            Procedure(String pWildcards),Long,Proc
Init 141                       Procedure(Long pFileAttributes)
ReadDirectories 142            Procedure(String pDir, Byte pShowWindow=False),Private     ! Recu
ResetFileCounters 142          Procedure(<String pDirectory>)
ScanDirectories 143            Procedure(<String pSD>, Byte pShowWindow=False, Byte pRecurse=Tru
ScanFiles 143                  Procedure(<String pDir>,<String pWC>, Long pAttrib=FF_:NORMAL, BY
SetFileFilter 145              Procedure(String pFF)
SetFileSort 145                Procedure(Byte pSort)
SetNoFileSort 146              Procedure
SetStartDir 146                Procedure(String pSD)
Construct 146                  Procedure
Destruct 147                   Procedure
```

3.11.4.1 CountFilesInDirectories

File Search Class - Methods

Prototype:	**(String pFF,<String pDirectory>),Long**
pFF	Wildcards that are used in the count. Multiple wildcards can be used, separated by semicolon (;). For example: '*.txt;*.php;*.css;*.js;*.htm;*.aspx'
[pDirectory]	Optional path to use. If it is not used then all directories are used.
Returns	Returns number of files counted

This method can be called after the ReadDirectories 142 method is called. It counts files in either all the directories read in by ReadDirectories 142, or just the one being passed in. Multiple wildcards can be used, separated by semicolon (;) For example '*.txt;*.php;*.css' would count all files with .txt, .php or .css extensions.

Example:

```
ITF  ITFileSearchClass
Fc   Long
 Code
 ITF.ReadDirectories('C:\Clarion',True)  !! Read all folders in C:\Clarion, showing
progress window
 Fc = ITF.CountFilesInDirectories('*.txt;*.php;*.css')  !! Count all .txt, .php and
.css files in the folders read.
```

See also:
ReadDirectories 142
ScanFiles 143
ResetFileCounters 142

3.11.4.2 GetFileSort

File Search Class - Methods

Prototype: **(), Byte**

Returns Returns the value of SELF.FileSort

This method returns the value of the FileSort 137 property. It determines the sort order used in ScanFiles.

Example:

```
ITF  ITFileSearchClass
FS   Byte
 Code
 FS = ITF.GetFileSort()
```

See also:
SetFileSort 145
SetNoFileSort 146
ScanFiles 143

3.11.4.3 GetLevel

File Search Class - Methods

Prototype: **(String pDir),Byte**

pDir Directory (folder) name to check.

Returns Returns the number of directory levels deep the folder is.

This method is used by the ReadDirectories 142 method to get the level of directories. The level is stored in the SubLevel field of the Directories 136 property and makes it easy to construct a directory tree as the level is already known. What this method does is simply loop through the pDir string and count the number of backslashes that are in it. It can also be used as standalone without any other

methods in this class being called.

Example:

```
ITF   ITFileSearchClass
P     CString(256)
Level Byte
 Code
 P = 'C:\Clarion\3rdParty\Template'
 Level = ITF.GetLevel(P)  !! Returns 3 as the "Template" is the third level from
the root (three '\' found)
```

See also:
Directories 136

ReadDirectories 142

3.11.4.4 GetWildcardList — File Search Class - Methods

Prototype: **(String pWildcards, Byte pNew=True),Long,Proc**

pWildcards	Wildcards to use. Multiple wildcards can be used, separated by semicolon (;). For example: '*.txt;*.php;*.css;*.js;*.htm;*.aspx'
pNew	Boolean flag that determines if the Wildcard queue is freed or not. By default the queue is freed when the method executes. By passing False in this parameter it is possible to call this method multiple times to add new wildcards to the list.
Returns	Number of wildcards in the Wildcards property

This methods takes a list of wildcards, separated by a semicolon (;) and adds each wildcard to the Wildcard property queue. At the end it returns the number of records in the Wildcard property queue. The wildcards are used in the CountFilesInDirectories 139 method.

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.GetWildCardList('*.txt;*.doc')
 ITS.GetWildCardList('*.clw;*.inc',False)  !! Add those also to the Wildcard
property
 ITS.GetWildCardList('*.txt;*.doc') !! This will reset the wildcards to only *.txt
and *.doc
```

See also:
CountFilesInDirectories 139

Construct 146

Destruct 147

3.11.4.5 Init — File Search Class - Methods

Prototype: **(Long pFileAttributes)**

pFileAttributes	File attributes to use when counting files in directories

Currently this method only calls the SetFileAttributes 144 method with the passed pFileAttributes. By default the FileAttributes 136 property is set to FF_:Normal.

Example:

```
ITF  ITFileSearchClass
 Code
 ITF.Init(FF_:Normal)
```

See also:
SetFileAttributes 144

FileAttributes 136

3.11.4.6 ReadDirectories

File Search Class - Methods

Prototype:	**(String pDir, Byte pShowWindow=False),Private**
pDir	Directory to start reading.
pShowWindow	Boolean that determines if a progress window is shown during the reading process.

This private method is called from ScanDirectories 143 to recursively read folder information for the pDir and all its subfolders. Note that it is only called from ScanDirectories 143 if the pRecurse parameter of ScanDirectories 143 is true. After ReadDirectories has been called, the Directories 136 property queue contains all the folders read. Note that the special "." and ".." folders are not included in the Directories queue.

See also:
ScanDirectories 143

3.11.4.7 ResetFileCounters

File Search Class - Methods

Prototype:	**(<String pDirectory>)**
[pDirectory]	Optional parameter which specifies directory to reset counters for. If omitted, file counters for all directories are set to zero.

This method is called from CountFilesInDirectories 139 and is used to reset the file counters for either the specified pDirectory directory or all directories in the Directories 136 property queue to zero. This would be called before repeated calls to ScanFiles 143 in case the file numbers have changed. An example could be some sort of file monitoring system that monitored the number of files in folders. Before repeatedly calling ScanFiles 143 you should call ResetFileCounters() as otherwise ScanFiles 143 will accumulate the number of files. Without the call to ResetFileCounters() in the example below, the ITF.Directories.FileCounter for each folder would double in the second call to the ScanFiles 143 method.

Example:

```
ITF  ITFileSearchClass
 Code
 ITF.ScanDirectories('C:\Temp')
```

```
ITF.ScanFiles(,,,True)   !! Count files
ITF.ResetFileCounters()  !! Reset
ITF.ScanFiles(,,,True)   !! Count files again
```

See also:
ScanDirectories 143
ScanFiles 143
Directories 136

3.11.4.8 ScanDirectories — File Search Class - Methods

Prototype: **(<String pSD>, Byte pShowWindow=False, Byte pRecurse=True),Long,Proc ! Returns number of directories**

[pSD] Start Directory. If omitted the current path is used.
pShowWindow Show progress window. Defaults to False
pRecurse Recurse through all subfolders. Defaults to True.

Returns Returns the number of directories.

This method reads in all folders in the selected path and optionally shows a progress window and reads all subfolders recursively. It calls the ReadDirectories 142 method to read through subfolders.

Example:
```
ITF  ITFileSearchClass
 Code
 ITS.ScanDirectories()
```

See also:
ReadDirectoriesx 142
TotalDirectories 138
StartDirectory 138
Directories 136

3.11.4.9 ScanFiles — File Search Class - Methods

Prototype: **(<String pDir>,<String pWC>, Long pAttrib=FF_:NORMAL, BYTE pCountOnly=False,BYTE pFreeFiles=True),Long,Proc**

[pDir] Omitable Folder to scan for files. If omitted all folders in SELF.Directories 136 will be processed.
[pWC] Omittable wildcard string. If omitted, *.* is used. If SELF.FileFilter 137 is specified it is used if this parameter is omitted or empty.
pAttrib Optional attribute for files to scan. Defaults to FF_:Normal. See DIRECTORY in the Clarion help for more information about attribute equates.
pCountOnly Optional parameter to indicate if the method should only count files and not

load them into the SELF.Files queue.

pFreeFiles Optional parameter to indicate if the method should free the SELF.Files queue. It defaults to True to preserve backward compatibility.

Returns Returns number of files scanned

This method scans for files with the provided parameters and either counts them, or adds them to the SELF.Files[137] queue. Before it is called, the ScanDirectories[143] must be called to load the directories. It is called from CountFilesInDirectories[139] to count files.

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.ScanDirectories()
 ITS.ScanFiles()
```

See also:
CountFilesInDirectories[139]

ScanDirectories[143]

Files[137]

3.11.4.10 SetFileAttributes

File Search Class - Methods

Prototype: **(Long pFileAttributes)**

pFileAttributes Attribute value

This method sets the value of the FileAttributes[136] property. It is used in CountFilesInDirectories[139] and ScanFiles[143] methods and initially set in the Construct[146] method to FF_:NORMAL. The FileAttributes[136] property determines the attribute for the files being read or counted in by ScanFiles[143].

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.SetFileAttributes(FF_:READONLY)  !! Scan only files that are read-only.
 ITS.ScanDirectories()
 ITS.ScanFiles()
```

See also:
ScanFiles[143]

ScanDirectories[143]

FileAttributes[519]

3.11.4.11 SetFileFilter

Prototype: **(String pFF)**

pFF File filter to assign to the FileFilter property

This method assigns a file filter to the FileFilter 137 property. The FileFilter 137 property is used in the ScanFiles 143 method to determine the wildcard to be used for the file search. It is set in the Construct 146 method to '*.*'

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.SetFileFilter('*.clw') !! Only search for *.clw files
 ITS.ScanDirectories()
 ITS.ScanFiles()
```

See also:

ScanFiles 143

Construct 146

FileFilter 137

3.11.4.12 SetFileSort

Prototype: **(Byte pSort)**

pSort Sort equate value

This method sets the file sort order that is used in the ScanFiles 143 method to sort the SELF.Files 137 queue after all the files have been read in. This makes it easy to set the file sort order before the files are scanned. Valid values are:

IT_FileSort:Name	Sort by file name
IT_FileSort:ShortName	Sort by short name (8.3 name)
IT_FileSort:Date	Sort by the file date
IT_FileSort:Time	Sort by the file time
IT_FileSort:Size	Sort by the file size
IT_FileSort:Attrib	Sort by the file attributes.

If SetFileSort is not called before scanning for files, then the sort order is undetermined.

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.SetFileSort(IT_FileSort:Name) !! Sort by name
 ITS.ScanDirectories()
 ITS.ScanFiles()
```

See also:

FileSort 137

ScanFiles 143

3.11.4.13 SetNoFileSort

File Search Class - Methods

Prototype: **(none)**

This method removes the file sort order that is used in the ScanFiles 143 method to sort the SELF.Files 137 queue after all the files have been read in. After this method has been called the Files queue will not be sorted at all.

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.SetFileSort(IT_FileSort:Name) !! Sort by name
 ITS.ScanDirectories()
 ITS.ScanFiles() !! ITS.Files queue is now sorted by name
 ITS.SetNoFileSort
 ITS.ScanFiles() !! ITS.Files queue is now unsorted.
```

See also:

Files 137

FileSort 137

SetFileSort 145

3.11.4.14 SetStartDir

File Search Class - Methods

Prototype: **(String pSD)**

pSD Start directory for ScanDirectories 143

This method sets the StartDirectory 138 property, which is set and used in the ScanDirectories 143 method.

Example:

```
ITF  ITFileSearchClass
 Code
 ITS.SetStartDir('C:\temp')
 ITS.ScanDirectories()
 ITS.ScanFiles() !! ITS.Files queue is now sorted by name
```

See also:

StartDirectory 138

ScanDireectories 143

3.11.4.15 Construct

File Search Class - Methods

Prototype: **(none)**

The Constructor in the File Search Class sets up 3 queues and assigns two properties. It creates the Directories 136, Files 137 and WildCard 139 queues and assigns '*.*' to FileFilter 137 and FF_:NORMAL to FileAttributes 136.

See also:
Destruct 147

3.11.4.16 Destruct

File Search Class - Methods

Prototype: **(none)**

The Destructor in the File Search Class frees and disposes of 3 queues, Directories 136, Files 137 and WildCard 139.

See also:
Construct 146

3.12 File Select Class

3.12.1 Overview File Select Class

The File Select Class is ues to select folders or files, i.e. prime and prompt using FileDialog. It can use variables or entry controls as default values.

```
ITFileSelectClass
Class(ITShellClass),TYPE,Module('ITFileSelectClass.clw'),Link('ITFileSelectClass',_
ITUtilLinkMode_),DLL(_ITUtilDllMode_)
FileMasks 152                  &ITFileMaskQueue                        ! Enumerated
list of filetypes
FileMask 152                   CString(2048),Private                   ! File mask
passed
FileName 152                   CString(FILE:MaxFileName),Private
FileFlags 151                  Long,Private
wCaption 153                   CString(256),Private
DefaultPath 151                CString(2049)
ForceDefaultPath 152           Byte
UseShortFileNames 153          Byte  !! Applies to file/folder names returned by
SelectDir and SelectFile

AddFileMask 154                Procedure(String pName, String pMask, Byte
pFree=False)  ! Adds a single mask
BuildFileMask 154              Procedure(),String
GetCaption 155                 Procedure(),String
GetFileMasks 155               Procedure(),String
GetFileName 156                Procedure(),String
Init 156                       Procedure(String pCaption, String pFileMasks, Long
pFlags, String pDefaultPath)
ParseFileMask 157              Procedure(String pFileMask, Byte pFree=True)     !
Parses file mask
SelectDir 157                  Procedure(*? pDirName),Long,Proc
SelectDir 157                  Procedure(Long pFEQ),Long,Proc
SelectFolder 158               Procedure(*? pDirName),Long,Proc ! Calls SelectDir
SelectFolder 158               Procedure(Long pFEQ),Long,Proc   ! Calls SelectDir
SelectFile 159                 Procedure(*CString pFileName, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SelectFile 159                 Procedure(*String pFileName, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SelectFile 159                 Procedure(Long pFEQ, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SetCaption 161                 Procedure(String pCaption)
SetDefaultDir 161              Procedure(String pDefaultPath)
SetDefaultFolder 162           Procedure(String pDefaultPath)
SetDefaultPath 163             Procedure(String pDefaultPath, Byte
pDefaultProgramPath=1)
SetFileMask 163                Procedure(String pFileMasks)    ! Calls ParseFileMask
SetFileName 164                Procedure(String pFileName)
SetForceDefaultpath 164        Procedure(Byte pForce=True)
SetUseLongNames 165            Procedure !!Sets UseShortFileNames to False
SetUseShortNames 166           Procedure !!Sets UseShortFileNames to True
Construct 166                  Procedure
Destruct 166                   Procedure
                           End
```

Example - Select a file:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask(154)('INI File','*.ini',True)
 FS.AddFileMask(154)('All Files','*.*.')
 FS.SetDefaultPath(163)(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetForceDefaultPath(164)()
 FS.SetCaption(161)('Select INI File') !! Set FileDialog caption
 FS.SelectFile(159)(?Loc:IniFileName)  !! Use control on window
 !!Or
 FS.SelectFile(159)(Loc:IniFileName)   !! Use variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

Example - Select a folder:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(163)(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetUseShortNames(166)             !! Force use of short filenames, use
SetUseLongNames(165) for long file names
 FS.SelectDir(157)(?Loc:IniFileName)  !! Use the FEQ from an entry field.
SelectFolder(158) is alternate method name
 !! Or:
 FS.SelectDir(157)(Loc:IniFileName)   !! Use a variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

Example - Build file mask string to use with FileDialog:

```
FS  ITFileSelectClass
F   CString(1025)
 Code
 FS.AddFileMask(154)('INI File','*.ini',True)  !! Start a new FileMask string
 FS.AddFileMask(154)('Text File','*.txt.')
 FS.AddFileMask(154)('CSV File','*.csv.')
 FS.AddFileMask(154)('All Files','*.*.')

 !! FS.BuildFileMask(154)() will now return the following string:
 !! 'INI File|*.ini|Text File|*.txt|CSV File|*.csv|All Files|*.*'

 If FileDialog('Select file',F,FS.BuildFileMask(154)(),FILE:KeepDir+FILE:LongName)
 End
```

Code example from Build Automator - Selecting DLLs:

```
FS  FileSelectClass
 Code
 FS.AddFileMask('DLL files (*.dll)','*.dll',True)
 If Loc:DLL
   FS.SetDefaultPath(Loc:DLL)
 Else
   FS.SetDefaultPath(INIMgr.Fetch('UpdateCallDLL','FS:DLLPath'))
 End
 FS.SelectFile(?Loc:DLL,False, False)
 INIMgr.Update('UpdateCallDLL','FS:DLLPath',FS.GetFilePart(Loc:DLL,FNS_FullPath))
```

Code example from Build Automator - Selecting MS-Build command line processor:

```
FS  FileSelectClass
 Code
 If Not Loc:MSBuildCommandLine
   FS.SetDefaultFolder(ITS.GetSpecialFolder(IT_CSIDL_WINDOWS) &
'\Microsoft.NET\Framework')
 End
 FS.AddFileMask('MSBuild', 'MSBuild.exe', True)  ! Adds a single mask
 FS.SelectFile(?Loc:MSBuildCommandLine)
```

Code example from Build Automator - Selecting and processing multiple file selections:

```
ITU  ITUtilityClass
FS   ITFileSelectClass
ITC  ITClarionClass
ITS  ITStringClass
 Code
 FS.AddFileMask('Clarion Application (*.app;*.prj)','*.app;*.prj',True)
 FS.SetDefaultPath(INIMgr.Fetch('UpdateClarionCompileFile','FS:Clarion
Application'))
 R = FS.SelectFile(Fn,True)
 If R
   R = ITU.MultiFileSelect(Clip(Fn))
   If R
     Sort(Loc:Applications,Loc:Applications.Loc:AppFile)
     LineN = Records(Loc:Applications)
     Loop I = R To 1 By -1
       Get(ITU.MSQ,I)
       Loc:Applications.Loc:AppFile = ITU.MultiFileSelPath & ITU.MSQ.MSFileN
       Get(Loc:Applications,Loc:Applications.Loc:AppFile)
       If ErrorCode()
         LineN += 1
         Clear(Loc:Applications)
         Loc:Applications.Loc:LineNr = LineN
         Loc:Applications.Loc:AppFile   = ITU.MultiFileSelPath & ITU.MSQ.MSFileN
         TargetFile = ITC.GetFilePart(Loc:Applications.Loc:AppFile,FNS_FullPath) &
'\' & ITC.GetTargetName(Loc:Applications.Loc:AppFile)
         If TargetFile
           Loc:Applications.Loc:DestFile =  TargetFile
         Else
           Loc:Applications.Loc:DestFile =  ''
         End
         Add(Loc:Applications,Loc:Applications.Loc:AppFile)  !! Add to maintain
filename sorting
       End
     End
     Sort(Loc:Applications,Loc:Applications.Loc:LineNr)
     INIMgr.update('UpdateClarionCompileFile','FS:Clarion
Application',ITU.MultiFileSelPath)
   End
 End
```

3.12.2 Data Types — File Select Class

The File Select Class has one datatype:

```
ITFileMaskQueue 151          QUEUE,TYPE
ITFileName                  CString(101)
ITFileMask                  CString(101)
                           END
```

3.12.2.1 ITFileMaskQueue

Queue to store masks for FileDialog(), i.e. what file types and wildcards.

```
ITFileMaskQueue          QUEUE,TYPE
ITFileName                 CString(101)
ITFileMask                 CString(101)
                         END
```

Store filename masks for FileDialog, for example 'Text Files|*.txt|All files|*.*' would be stored as two records in the derived queue.

3.12.3 Properties

There are currently 8 properties in the File Select Class:

```
FileMasks 152              &ITFileMaskQueue                    ! Enumerated
list of filetypes
FileMask 152               CString(2048),Private               ! File mask
passed
FileName 152               CString(FILE:MaxFileName),Private
FileFlags 151              Long,Private
wCaption 153               CString(256),Private
DefaultPath 151            CString(2049)
ForceDefaultPath 152       Byte
UseShortFileNames 153      Byte  !! Applies to file/folder names returned by
SelectDir and SelectFile
```

3.12.3.1 DefaultPath

The **DefaultPath** property is used in the SetDefaultPath 163, SetDefaultFolder 162, SetDefaultDir 161, SelectFile 159 and SelectDir 157 methods and is used to store the default path to use. If ForceDefaultPath 152 property is set to TRUE then the DefaultPath is forced as the default path when the FileDialog is opened in the SelectFile 159 method. If ForceDefaultPath 152 is set to FALSE then the DefaultPath is only used if no path is passed to SelectFile 159. Note that ForceDefaultPath 152 does not currently apply to the SelectDir 157 method.

It is declared as:

```
DefaultPath                CString(2049)
```

3.12.3.2 FileFlags

The **FileFlags** property is currently not used, but reserved for use later.

It is declared as:

```
FileFlags                  Long,Private
```

3.12.3.3 FileMask

The **FileMask** property is used in the GetFileMasks 155, SetFileMasks 164 and BuildFileMask 154 methods and is used to store the full file mask string, for example: 'Text Files|*.txt|All files|*.*' SetFileMasks 164 calls the ParseFileMask 157 which parses the string passed in and in turns calls AddFileMask 154 to add the file mask to the FileMasks 152 queue.

It is declared as:

```
FileMask                CString(2048),Private                ! File mask
passed
```

3.12.3.4 FileMasks

The **FileMasks** property queue is used in the AddFileMask 154, BuildFileMask 154 methods and is used to store a list of filemasks broken up into the name and associated wildcard. For example, the file mask of 'Text Files|*.txt|All files|*.*' would be stored in two entries:

Name	Wildcard
Text Files	*.txt
All files	*.*

It is declared as:

```
FileMasks               &ITFileMaskQueue                     ! Enumerated
list of filetypes
```

3.12.3.5 FileName

The **FileName** property is currently not used, but reserved for use later.

It is declared as:

```
FileName                CString(FILE:MaxFileName),Private
```

3.12.3.6 ForceDefaultPath

The **ForceDefaultPath** property is used in the SetForceDefaultPath 164 and SelectFile 159 methods and is used to determine if the DefaultPath 151 should be forced when the FileDialog is opened rather than the current value of the variable or control that is being used to select. This means that the start folder for the FileDialog will always be set to DefaultPath 151, no matter what value is in the variable or the entry control for the selection. If DefaultPath 151 has **not** been set when the SelectFile 159 method is called, then SelectFile 159 will behave as if **ForceDefaultPath** is False.

It is declared as:

```
ForceDefaultPath        Byte
```

3.12.3.7 UseShortFileNames

The **UseShortFilename** property is used in the SetUseShortNames 166, SetUseLongNames 165, SelectFile 159, SelectDir 157 and Construct 166 methods and is used to set the filename to either long or short with LongPath and ShortPath respectively before it is returned from SelectFile 159 and SelectDir 157.

It is declared as:

```
UseShortFileNames           Byte
```

3.12.3.8 wCaption

The **wCaption** property is used in the GetCaption 155, SetCaption 161, SelectFile 159 and SelectDir 157 methods and is used to set the caption for the FileDialog when called from the SelectFile 159 and SelectDir 157 methods.

It is declared as:

```
wCaption                    CString(256),Private
```

3.12.4 Methods — File Select Class

There are currently 25 methods in the File Select Class.

```
AddFileMask 154                 Procedure(String pName, String pMask, Byte
pFree=False)  ! Adds a single mask
BuildFileMask 154               Procedure(),String
GetCaption 155                  Procedure(),String
GetFileMasks 155                Procedure(),String
GetFileName 156                 Procedure(),String
Init 156                        Procedure(String pCaption, String pFileMasks, Long
pFlags, String pDefaultPath)
ParseFileMask 157               Procedure(String pFileMask, Byte pFree=True)     !
Parses file mask
SelectDir 157                   Procedure(*? pDirName),Long,Proc
SelectDir 157                   Procedure(Long pFEQ),Long,Proc
SelectFolder 158                Procedure(*? pDirName),Long,Proc ! Calls SelectDir
SelectFolder 158                Procedure(Long pFEQ),Long,Proc   ! Calls SelectDir
SelectFile 159                  Procedure(*CString pFileName, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SelectFile 159                  Procedure(*String pFileName, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SelectFile 159                  Procedure(Long pFEQ, Byte pMulti=False, Byte
pSave=False, Byte pSuggestFileName=False),BYTE,PROC
SetCaption 161                  Procedure(String pCaption)
SetDefaultDir 161               Procedure(String pDefaultPath)
SetDefaultFolder 162            Procedure(String pDefaultPath)
SetDefaultPath 163              Procedure(String pDefaultPath, Byte
pDefaultProgramPath=1)
SetFileMask 163                 Procedure(String pFileMasks)    ! Calls ParseFileMask
SetFileName 164                 Procedure(String pFileName)
SetForceDefaultpath 164         Procedure(Byte pForce=True)
SetUseLongNames 165             Procedure !!Sets UseShortFileNames to False
SetUseShortNames 166            Procedure !!Sets UseShortFileNames to True
```

```
Construct 166          Procedure
Destruct 166           Procedure
```

3.12.4.1 AddFileMask

Prototype: **(String pName, String pMask, Byte pFree=False)**

pName	Descriptive name for the file mask, such as "Text files" or "All files"
pMask	Wildcard mask for the file type, such as "*.txt" or "*.*"
pFree	Specifies if existing entries should be deleted. Set to True if the filemask needs to be started over and the existing entries removed.

This method adds a single entry into the file mask queue. The file mask is used to construct a string that is used for the *extensions* parameter when FileDialog() is called by the SelectFile 159 method.

Example:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask('INI File','*.ini',True)
 FS.AddFileMask('All Files','*.*.')
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SelectFile(?Loc:IniFileName)

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
BuildFileMask 154
GetFileMask 155
SetFileMask 163
FileMasks 152
FileMask 152
ITFileMaskQueue 151

3.12.4.2 BuildFileMask

Prototype: **(),String**

Returns Returns a string with the filemask 152 to use with FileDialog()

This method loops through the FileMasks 152 property queue and builds a string with the file masks, for example 'Text Files|*.txt|All files|*.*' This method is called by the SelectFile 159 method and you do not need to call it unless you want to use the FileMask to keep track of your own file masks, i.e. without using the SelectFile 159 method.

Example:

```
FS  ITFileSelectClass
```

```
F   CString(1025)
 Code
 FS.AddFileMask('INI File','*.ini',True)  !! Start a new FileMask string
 FS.AddFileMask('Text File','*.txt.')
 FS.AddFileMask('CSV File','*.csv.')
 FS.AddFileMask('All Files','*.*.')

 !! FS.BuildFileMask() will now return the following string:
 !! 'INI File|*.ini|Text File|*.txt|CSV File|*.csv|All Files|*.*'

 If FileDialog('Select file',F,FS.BuildFileMask(),FILE:KeepDir+FILE:LongName)
 End
```

See also:
AddFileMask 154

FileMask 152

FileMasks 152

3.12.4.3 GetCaption — File Select Class - Methods

Prototype: **(),String**

Returns Returns the contents of the wCaption 88 property

This method returns the contents of the wCaption 153 property. The wCaption property stores the caption to be used with the FileDialog call in the SelectFile 159 and SelectDir 157 methods. wCaption defaults to "Select File" and "Select Folder" for SelectFile 159 and SelectFolder 158/SelectDir 157 respectively.

Example:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask('INI File','*.ini',True)
 FS.AddFileMask('All Files','*.*.')
 FS.SetCaption('Select INI file')
 FS.SelectFile(?Loc:IniFileName)
 Message('Caption was: ' & FS.GetCaption())  !! Displays "Select INI file"
```

See also:
SetCaption 161

SelectDir 157

SelectFolder 158

SelectFile 159

wCaption 153

3.12.4.4 GetFileMasks — File Select Class - Methods

Prototype: **(),String**

Returns Returns the contents of the FileMask 152 string property.

This method returns the contents of the FileMask 152 property. The FileMask 152 property is a string that is passed in by the SetFileMask 163 method. SetFileMask 163 method calls the ParseFileMask 157 method which breaks it down and adds it to the FileMasks 152 queue property.

Example:

```
FS  ITFileSelectClass
F   CString(1025)
 Code
 FS.SetFileMask('INI File|*.ini|Text File|*.txt|CSV File|*.csv|All Files|*.*')
 !! Same as calling:
 !! FS.AddFileMask('INI File','*.ini',True)  !! Start a new FileMask string
 !! FS.AddFileMask('Text File','*.txt.')
 !! FS.AddFileMask('CSV File','*.csv.')
 !! FS.AddFileMask('All Files','*.*.')
 F = FS.GetFileMasks()  !! F now contains: 'INI File|*.ini|Text File|*.txt|CSV
File|*.csv|All Files|*.*'
```

See also:
SetFileMask 163

ParseFileMask 157

SelectFile 159

FileMask 152

FileMasks 152

3.12.4.5 GetFileName

Prototype: **(),String**

Returns Return an empty string

This method is currently not used. Method is reserved for future use.

See also:
SetFileName 164

3.12.4.6 Init

Prototype: **(String pCaption, String pFileMasks, Long pFlags, String pDefaultPath)**

pCaption

pFileMasks

pFlags

pDefaultPath

This method is currently not used. Method is reserved for future use.

See also:

3.12.4.7 ParseFileMask

Prototype: **(String pFileMask, Byte pFree=True)**

pFileMask File mask to parse

pFree Indicates if it should free the FileMasks queue property before adding to it.

This method takes a file mask, for example 'Text Files|*.txt|All files|*.*' and parses it to add it to the FileMasks 152 queue property. Normally you do not call it directly but rather would call SetFileMask 163 method which also sets the FileMask 152 string property. ParseFileMask may be set as private in a future release.

Example:

```
FS  ITFileSelectClass
F   CString(1025)
 Code
 FS.ParseFileMask('INI File|*.ini|Text File|*.txt|CSV File|*.csv|All Files|*.*')

 !! Note that at this point FS.FileMask is still an empty string!

 !! Same as calling:
 !! FS.AddFileMask('INI File','*.ini',True)  !! Start a new FileMask string
 !! FS.AddFileMask('Text File','*.txt.')
 !! FS.AddFileMask('CSV File','*.csv.')
 !! FS.AddFileMask('All Files','*.*.')
```

See also:

GetFileMasks 155

SetFileMask 163

SelectFile 159

FileMask 152

FileMasks 152

3.12.4.8 SelectDir

Prototype: **(*? pDirName),Long,Proc**

Overloaded:

(Long pFEQ),Long,Proc

pDirName Variable reference containing the variable that contains and should receive the newly selected folder name

pFEQ Control reference (FEQ) pointing to a control on a window that contains and should receive the newly selected folder name

Returns Returns the value returned by the call to FileDialog: "FILEDIALOG returns zero

(0) if the user pressed the Cancel button, or one (1) if the user pressed the Ok button on the file choice dialog."

This method puts it all together and calls FileDialog with the settings that have been set up. This method comes in two flavors. One where you pass it a variable that receives the new selected folder name. The variable can be any appropriate datatype, such as a string or cstring. The other uses a control label (FEQ) that it will set to the new selected folder name.

If the control or variable contains information (existing filename information) the SelectDir will check if the information passed in is a folder name or file name and use the folder name part of it to construct the default folder where it will open to.

If no path information passed in and the DefaultPath 151 property is set then it will be used unconditionally as the startup folder.

If no path information is passed in and the DefaultPath 151 property is **not** set then the current path (Path()) will be used.

SelectDir does not use any calls to SetPath() so it never changes the currently used path!

The UseShortFileNames 153 property is used to determine if the returned folder name is short or long. By default the method will return the **long** folder name.

In the example below the default path is retrieved and stored using the INI Manager. This way you can make the SelectDir default to whatever directory was used last time.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetUseShortNames
 FS.SelectDir(?Loc:IniFileName)  !! Use the FEQ from an entry field
 !! Or:
 FS.SelectDir(Loc:IniFileName)   !! Use a variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:

SelectFolder 158

SetDefaultDir 161

SetDefaultFolder 162

SetUseShortNames 166

SetUseLongNames 165

UseShortFileNames 153

3.12.4.9 SelectFolder

File Select Class - Methods

Prototype: **(*? pDirName),Long,Proc**

Overloaded:

(Long pFEQ),Long,Proc

pDirName Variable reference containing the variable that contains and should receive the newly selected folder name

pFEQ Control reference (FEQ) pointing to a control on a window that contains and

should receive the newly selected folder name

Returns Returns the value returned by the call to FileDialog: "FILEDIALOG returns zero (0) if the user pressed the Cancel button, or one (1) if the user pressed the Ok button on the file choice dialog."

This method puts it all together and calls FileDialog with the settings that have been set up. This method comes in two flavors. One where you pass it a variable that receives the new selected folder name. The variable can be any appropriate datatype, such as a string or cstring. The other uses a control label (FEQ) that it will set to the new selected folder name.

If the control or variable contains information (existing filename information) the SelectFolder will check if the information passed in is a folder name or file name and use the folder name part of it to construct the default folder where it will open to.

If no path information passed in and the DefaultPath 151 property is set then it will be used unconditionally as the startup folder.

If no path information is passed in and the DefaultPath 151 property is **not** set then the current path (Path()) will be used.

SelectFolder does not use any calls to SetPath() so it never changes the currently used path!

The UseShortFileNames 153 property is used to determine if the returned folder name is short or long. By default the method will return the **long** folder name.

In the example below the default path is retrieved and stored using the INI Manager. This way you can make the SelectFolder default to whatever directory was used last time.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetUseShortNames
 FS.SelectFolder(?Loc:IniFileName)  !! Use the FEQ from an entry field
 !! Or:
 FS.SelectFolder(Loc:IniFileName)   !! Use a variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:

SelectDir 157

SetDefaultDir 161

SetDefaultFolder 162

SetUseShortNames 166

SetUseLongNames 165

UseShortFileNames 153

3.12.4.10 SelectFile

File Select Class - Methods

Prototype: **(*CString pFileName, Byte pMulti=False, Byte pSave=False, Byte pSuggestFileName=False),BYTE,PROC**

Overloaded:

(*String pFileName, Byte pMulti=False, Byte pSave=False, Byte

pSuggestFileName=False),BYTE,PROC

Overloaded:

(Long pFEQ, Byte pMulti=False, Byte pSave=False, Byte pSuggestFileName=False),BYTE,PROC

pFileName	String or CString variable that may contain existing information about file name
pFEQ	Windows control label (FEQ) that may contain existing information about file name
pMulti	Determines if the FileDialog opened in OPEN mode can allow selecting multiple files. Not applicable in SAVE mode (pSave=True)
pSave	Determines if the FileDialog is opened in OPEN or SAVE mode. Default is OPEN.
pSuggestFileName	Determines if FileDialog suggests the name of the file name passed in as pFileName. If False, then only the path name will be used.
Returns	Returns the value returned by the call to FileDialog: "FILEDIALOG returns zero (0) if the user pressed the Cancel button, or one (1) if the user pressed the Ok button on the file choice dialog."

This method puts it all together and calls FileDialog with the settings that have been set up. This method comes in two flavors. One where you pass it a variable that receives the new selected folder name. The variable can be any appropriate datatype, such as a string or cstring. The other uses a control label (FEQ) that it will set to the new selected folder name.

If the control or variable contains information (existing filename information) the SelectFile will check if the information passed in is a folder name or file name and use the folder name part of it to construct the default folder where it will open to.

If ForceDefaultPath 152 is True and DefaultPath 151 contains a path, then the DefaultPath 151 will be unconditionally used as a startup folder for the file selection.

If DefaultPath 151 contains a path it will be used only if the pFileName or pFEQ is empty - i.e. does not contain file name.

If DefaultPath 151 is empty and pFilename/pFEQ is also empty then Path() will be used as a startup folder.

SelectFile does not use any calls to SetPath() so it never changes the currently used path!

The UseShortFileNames 153 property is used to determine if the returned file name is short or long. By default the method will return the **long** file name.

Example:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask('INI File','*.ini',True)
 FS.AddFileMask('All Files','*.*.')
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetForceDefaultPath()
 FS.SetCaption('Select INI File') !! Set FileDialog caption
 FS.SelectFile(?Loc:IniFileName)  !! Use control on window
 !!Or
 FS.SelectFile(Loc:IniFileName)   !! Use variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:

SelectFolder 158
SetCaption 161
AddFileMask 154
SetFileMask 163
SetForceDefaultpath 164
SetUseLongFileNames 165
SetUseShortFileNames 166

3.12.4.11 SetCaption

File Select Class - Methods

Prototype: **(String pCaption)**

pCaption Caption to use when FileDialog is called from SelectFile and SelectFolder/SelectDir

This method sets the wCaption 153 property that is used when FileDialog() is called. wCaption 153 defaults to "Select File" and "Select Folder" for SelectFile 159 and SelectFolder 158/SelectDir 157 respectively.

Example:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask('INI File','*.ini',True)
 FS.AddFileMask('All Files','*.*.')
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetForceDefaultPath()
 FS.SetCaption('Select INI File') !! Set FileDialog caption
 FS.SelectFile(?Loc:IniFileName)  !! Use control on window
 !!Or
 FS.SelectFile(Loc:IniFileName)   !! Use variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
GetCaption 155
SelectDir 157
SelectFolder 158
SelectFile 159
wCaption 153

3.12.4.12 SetDefaultDir

File Select Class - Methods

Prototype: **(String pDefaultPath)**

pDefaultPath Path to assign to DefaultPath 151.

This method calls the SetDefaultPath 163 to assign the DefaultPath 151 property with the pDefaultPath parameter. For more detailed information see SetDefaultPath 163.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultDir(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename')) !! Set the
default path
 FS.SetUseShortNames
 FS.SelectDir(?Loc:IniFileName)

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:

SetDefaultFolder 162

SetDefaultPath 163

SelectFile 159

SelectFolder 158

SelectDir 157

SetForceDefaultPath 164

DefaultPath 151

ForceDefaultPath 152

3.12.4.13 SetDefaultFolder

File Select Class - Methods

Prototype: **(String pDefaultPath)**

pDefaultPath Path to assign to DefaultPath 151.

This method calls the SetDefaultPath 163 to assign the DefaultPath 151 property with the pDefaultPath parameter. For more detailed information see SetDefaultPath 163.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultFolder(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename')) !! Set the
default path
 FS.SetUseShortNames
 FS.SelectDir(?Loc:IniFileName)

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:

SetDefaultFolder 162

SetDefaultPath 163

SelectFile 159

SelectFolder 158

SelectDir 157
SetForceDefaultPath 164
DefaultPath 151
ForceDefaultPath 152

3.12.4.14 SetDefaultPath

File Select Class - Methods

Prototype: **(String pDefaultPath, Byte pDefaultProgramPath=1)**

pDefaultPath Path to assign to DefaultPath 151.

pDefaultProgramPath Determines if the default program path should be used if pDefaultPath is empty

This method sets the DefaultPath 151 string property. If pDefaultProgramPath is true (default) the DefaultPath 151 will be set to the program path (SELF.ProgPath 56) The DefaultPath 151 property is used in the SelectFile 159 and SelectFolder 158/SelectDir 157 methods to set a start folder when FileDialog() is called.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename')) !! Set the
default path
 FS.SetUseShortNames
 FS.SelectDir(?Loc:IniFileName)

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
SetDefaultFolder 162
SetDefaultDir 161
SelectFile 159
SelectFolder 158
SelectDir 157
SetForceDefaultPath 164
DefaultPath 151
ForceDefaultPath 152

3.12.4.15 SetFileMask

File Select Class - Methods

Prototype: **(String pFileMasks)**

pFileMasks File mask to use.

This method sets the FileMask 152 string property and then calls the ParseFileMask 157 method to parse the file mask into entries for the FileMasks 152 queue property. The pFileMask parameter must contain valid file masks in the format expected by FileDialog(), i.e. "File name|WildCard" for example 'Text Files|*.txt|All files|*.*'

Example:

```
FS  ITFileSelectClass
F   CString(1025)
 Code
 FS.SetFileMask('INI File|*.ini|Text File|*.txt|CSV File|*.csv|All Files|*.*')
 !! Same as calling:
 !! FS.AddFileMask('INI File','*.ini',True)  !! Start a new FileMask string
 !! FS.AddFileMask('Text File','*.txt.')
 !! FS.AddFileMask('CSV File','*.csv.')
 !! FS.AddFileMask('All Files','*.*.')
 F = FS.GetFileMasks()  !! F now contains: 'INI File|*.ini|Text File|*.txt|CSV
File|*.csv|All Files|*.*'
```

See also:
GetFileMasks 155
AddFileMask 154
BuildFileMask 154
ParseFileMask 157
SelectFile 159
FileMask 152
FileMasks 152

3.12.4.16 SetFileName

Prototype: **(String pFileName)**

pFileName Parameter not used.

This method is currently not used. Method is reserved for future use.

See also:
GetFileName 156

3.12.4.17 SetForceDefaultpath

Prototype: **(Byte pForce=True)**

pForce Determines if ForceDefaultPath is set to True or False. Defaults to True.

This method assigns the value of pForce to the ForceDefaultPath 152 property. pForce defaults to True. If ForceDefaultPath 152 is set to True then it will be forced as the startup folder when SelectFile 159 or SelectFolder 158/SelectDir 157 is called, unless the DefaultPath 151 property is empty. This gives you the option to force the FileDialog to open for example in the last used folder, as in the example below, where the folder is stored using the INI Manager class (ABC)

Example:

```
FS  ITFileSelectClass
 Code
 FS.AddFileMask('INI File','*.ini',True)
 FS.AddFileMask('All Files','*.*')
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetForceDefaultPath()         !! Force use of DefaultPath
 FS.SetCaption('Select INI File') !! Set FileDialog caption
 FS.SelectFile(?Loc:IniFileName)  !! Use control on window
 !!Or
 FS.SelectFile(Loc:IniFileName)   !! Use variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
SetDefaultDir 161

SetDefaultFolder 162

SetDefaultPath 163

ForceDefaultPath 152

DefaultPath 151

3.12.4.18 SetUseLongNames

Prototype: **(none)**

This method sets the UseShortFileNames 153 to False. This instructs SelectFile 159 and SelectFolder 158/SelectDir 157 to return the long file- or path name.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetUseLongNames             !! Force use of Long file/folder names
 FS.SelectDir(?Loc:IniFileName)  !! Use the FEQ from an entry field
 !! Or:
 FS.SelectDir(Loc:IniFileName)   !! Use a variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
SetUseShortNames 166

SelectFile 159

SelectFolder 158

SelectDir 157

UseShortFileNames 153

3.12.4.19 SetUseShortNames

Prototype: **(none)**

This method sets the UseShortFileNames 153 to True. This instructs SelectFile 159 and SelectFolder 158/SelectDir 157 to return the short file- or path name.

Example:

```
FS  ITFileSelectClass
 Code
 FS.SetDefaultPath(INIMgr.Fetch('UpdateGetFromINI','FS:INIFilename'))
 FS.SetUseShortNames             !! Force use of short file/folder names
 FS.SelectDir(?Loc:IniFileName)  !! Use the FEQ from an entry field
 !! Or:
 FS.SelectDir(Loc:IniFileName)   !! Use a variable

INIMgr.Update('UpdateGetFromINI','FS:INIFilename',FS.GetFilePart(Loc:IniFileName,FN
S_FullPath))
```

See also:
SetUseLongNames 165
SelectFile 159
SelectFolder 158
SelectDir 157
UseShortFileNames 153

3.12.4.20 Construct

Prototype: **(none)**

The Constructor in the File Select Class sets up 1 queue property and assigns one boolean property. It creates the FileMasks 152 queue property and sets the UseShortFileNames 153 to False making the class default to returning Long file names from SelectFile 159 and SelectFolder 158/SelectDir 157 methods.

See also:
Destruct 166

3.12.4.21 Destruct

Prototype: **(none)**

The Destructor in the File Select Class frees and disposes of 1 queue property, the FileMasks 152.

See also:
Construct 166

3.13 Files Class

3.13.1 Overview — Files Class

This class currently has just one method in it. Pass a file lable to the GetFilePrefix and you will get back the prefix string for the file. Very useful when using PROP:Alias to set the prefix for SQL statements.

```
ITFilesClass            CLASS(ITStringClass 280
),TYPE,Module('ITFilesClass.clw'),Link('ITFilesClass',_ITUtilLinkMode_),DLL(_ITUtil
DllMode_)
GetFilePrefix 167           Procedure(FILE pF),String
                        END
```

3.13.2 Methods — Files Class

There is currently just one method in the Files Class 167.

```
GetFilePrefix 167           Procedure(FILE pF),String
```

3.13.2.1 GetFilePrefix — Files Class - Methods

Prototype: **(FILE pF),String**

pF Reference to a file structure.

Returns Returns the prefix of the file structure.

This method retrieves the prefix of the file by getting the label of the first field in the file structure and then calling the GetFieldPrefix 299 method in the String Class 280 to parse out the prefix. Note that the prefix is returned without the trailing colon (:)

Example:

```
ITF  ITFilesClass
F       File,Driver(ASCII),PRE(ASCII),CREATE
Record    Record
F1          String(255)
          End
        End

 Code
 Message('The file prefix is: ' & ITF.GetFilePrefix(F))
```

See also:
GetFieldPrefix 299

3.14 Global Thread Class

3.14.1 Overview Global Thread Class

```
ITGlobalThreadClass
CLASS(ITUtilityClass),TYPE,Module('ITGlobalThreadClass.clw'),Link('ITGlobalThreadCl
ass',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
CriticalSection           &ICriticalSection,PRIVATE
WindowThreads             &ITGlobalThreadQ
FrameWindow               &Window
FrameProcedure            CString(256)
FrameThread               Long
AddWindow                 Procedure(WINDOW pWin, String pProcedureName, Byte
pIsFrame=0)
GetInsertLevel            Procedure(Long pThread),Long
CloseWindow               Procedure(Long pThread, Long pHandle)
CloseAllWindows           Procedure
RemoveWindow              Procedure
Construct                 Procedure
Destruct                  Procedure
                        END
```

3.14.2 Properties Global Thread Class

Enter topic text here.

3.14.3 Methods Global Thread Class

Enter topic text here.

3.15 Hyperlink Class

3.15.1 Overview Hyperlink Class

```
ITHyperLinkClass
Class(ITFileClass),TYPE,Module('ITHyperLinkClass.clw'),Link('ITHyperLinkClass',_ITU
tilLinkMode_),DLL(_ITUtilDllMode_)
HyperLinks                  &ITControlQ
Accepted                    Long  ! Ctrl accepted
CursorFile                  CString(256)
DefaultTip                  CString(256)
UseDefaultTip               Byte  ! Set before calling RegisterControl
Initialized                 Byte

Init                        Procedure(<String pCursor>,<String pTip>)
Kill                        Procedure
RegisterControl             Procedure(Long pFEQ, Long pContentsFEQ=0, Byte
pChangeToString=True, <String pCursor>,<String pURL>)
UnRegisterControl           Procedure(Long pFEQ)
HyperLinkAccepted           Procedure(Long pFEQ),Byte ! Returns true if pFEQ is
found in HyperLinks
TakeAccepted                Procedure(),BYTE,VIRTUAL  ! Event handler for regions
TakeResize                  Procedure(),BYTE,PROC!,PRIVATE  ! Event handler for
resizing
SetControlPositions         Procedure(Long pCtrl, Long pRgn)
SetControlFonts             Procedure(Long pCtrl, Long pString)
SetHyperLink                Procedure(Long pFEQ)
SetLinkTip                  Procedure(Long pFEQ, String pTip, Byte
pAppendContents=TRUE)
HyperLinkExists             Procedure(Long pFEQ),Byte
Construct                   Procedure
Destruct                    Procedure
                          End
```

3.15.2 Properties Hyperlink Class

Enter topic text here.

3.15.3 Methods Hyperlink Class

Enter topic text here.

3.16 INI Class

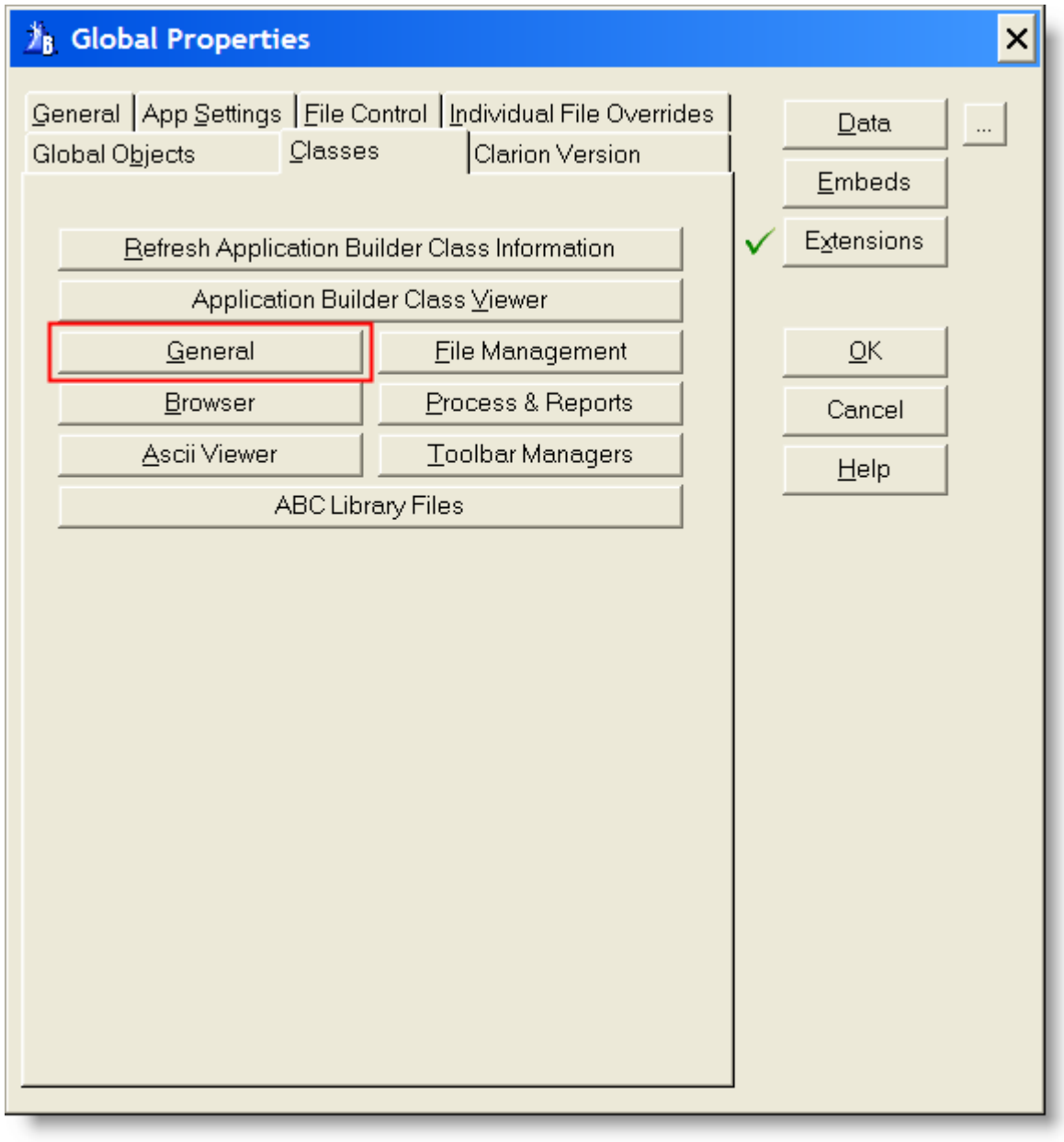
3.16.1 Overview INI Class

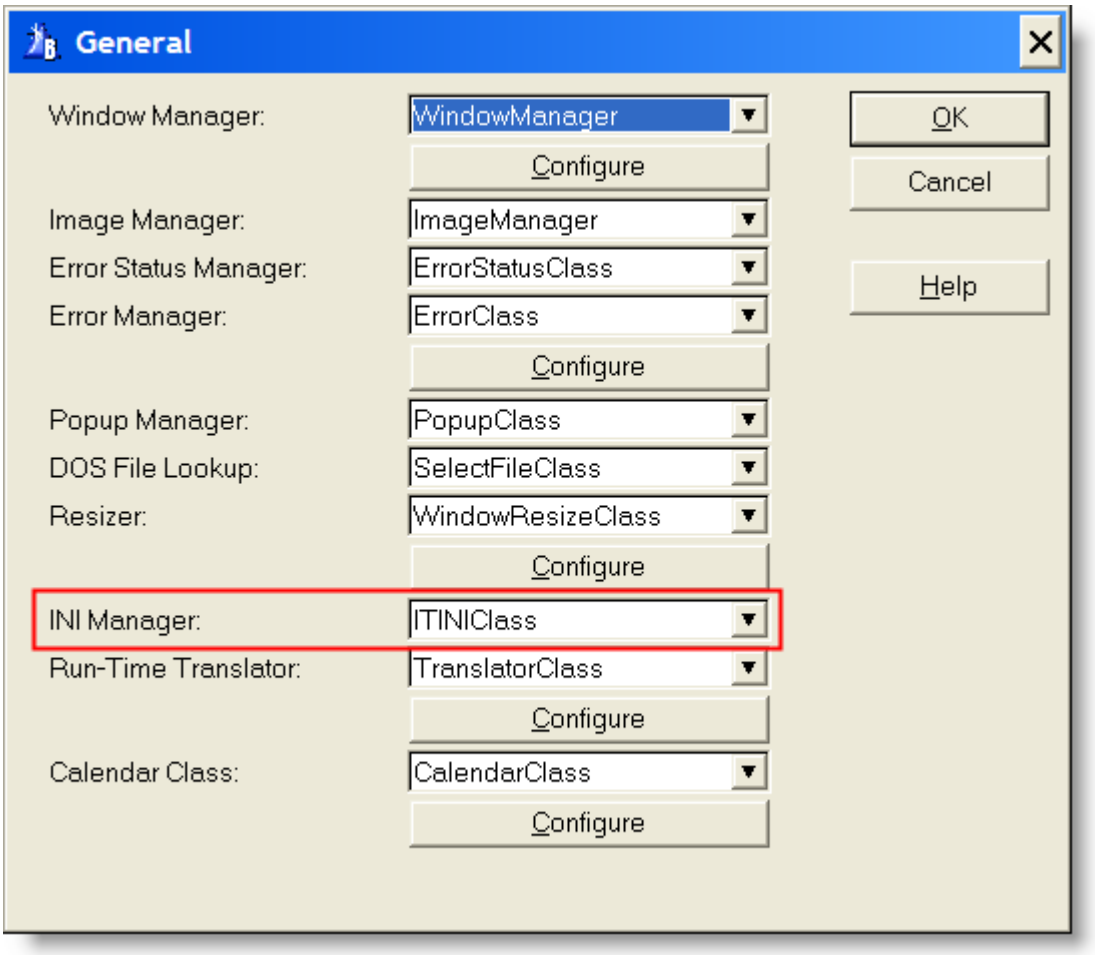
The INI Class is derived from the INIClass and only contains two methods that override the Fetch and Kill methods.

```
ITINIClass                CLASS(INIClass),TYPE,Module('ITINIClass.clw'),Link
('ITINIClass',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
Kill 173                      Procedure
Fetch 172                     Procedure(STRING Sector,STRING Name),STRING
Update 173                    Procedure(STRING ProcName,WINDOW W)
                          END
```

Implementing the Class:

After Refreshing ABC classes, go to "Application | Global Properties" from the Clarion IDE main menu. Then click on the "Classes" tab and then the "General" button.





In the "INI Manager" drop down select **ITINIClass** instead of INIClass and you have activated the **ITINIClass** as the main INI class.

3.16.2 Methods — INI Class

The INI class has 3 methods.

Kill 173	`Procedure`
Fetch 172	`Procedure(STRING Sector,STRING Name),STRING`
Update 173	`Procedure(STRING ProcName,WINDOW W)`

3.16.2.1 Fetch — INI Class - Methods

Prototype: **(STRING Sector,STRING Name),STRING**

Sector Name of sector in INI file
Name Entry name in INI file

Returns String with information from INI or empty string if entry does not exist

This method overrides the Fetch method so that it returns an empty string instead of the value of the INIUnknown constant which is defined in ABUTIL.CLW as:

```
INIUnknown      EQUATE('---Unknown---')
```

With the standard method you will have to check for INIUnknown rather than an empty string.

3.16.2.2 Kill

Prototype: **(none)**

This method overrides the Kill method to get rid of the '__Dont_Touch_Me__' section entries in your INI files and your registry. Those entries look very unprofessional and do not serve any purpose as far as we have been able to determine.

3.16.2.3 Update

Prototype: **(STRING ProcName,WINDOW W)**

ProcName	Name of the procedure
W	Window reference

This method overrides the Update method in the ABC INIClass to allow the application frame to open on the monitor where it was closed on. The default settings in the ABC INI class is that if the application frame is maximized the X and Y position isn't written to the INI file. When the application frame opens again it will therefore default to being at X,Y screen coordinates 0,0, which will normally be on the primary monitor.

Use this class along with the Window Fixer product to make sure that your application frame opens up on the correct monitor.

See also:
Window Fixer

3.17 Image Class

3.17.1 Overview Image Class

```
ITImageClass
Class(ITFileClass),TYPE,Module('ITImageClass.clw'),Link('ITImageClass',_ITUtilLinkM
ode_),DLL(_ITUtilDllMode_)
BufferValue                Short,Private
UseBufferValue             Byte
BufferSet                  Byte

SetBufferValue             Procedure(Short pBuffer)
GetBufferValue             Procedure(),Short
UseBufferValue             Procedure(Byte pUseBuffer)
CheckBuffer                Procedure
ResizeImage                Procedure(Long pFEQ, String pImage,<Long pW>, <Long pH>)
GetImageSize               Procedure(Long pImageFEQ, *Long pW, *Long pH, Byte
pPixels=False)
SetImageSize               Procedure(Long pImageFEQ, Long pW, Long pH, Byte
pPixels=False)
GetSetImageSize            Procedure(Long pImageFEQ, Byte pPixels=False)
SetRelativePosition        Procedure(Long pImageFEQ, Long pRelativeFEQ, Long pX,
Long pY, Byte pJust)
Construct                  Procedure
Destruct                   Procedure
                         End
```

3.18 Keyboard Class

3.18.1 Properties Keyboard Class

Enter topic text here.

3.18.2 Methods Keyboard Class

Enter topic text here.

3.18.3 Overview Keyboard Class

Enter topic text here.

3.19 Locale Class

3.19.1 Overview Locale Class

```
ITLocaleClass
CLASS(ITUtilityClass),TYPE,Module('ITLocaleClass.clw'),Link('ITLocaleClass',_ITUtil
LinkMode_),DLL(_ITUtilDllMode_)
MonthNames                CString(41),Dim(12)
MonthsFilled              Byte
GetMonths                 Procedure
GetMonthName              Procedure(Byte pMonth),String
Construct                 Procedure
Destruct                  Procedure
                        END
```

3.19.2 Properties Locale Class

Enter topic text here.

3.19.3 Methods Locale Class

Enter topic text here.

3.20 Macro Class

3.20.1 Overview Macro Class

```
ITMacroClass
Class(ITUtilityClass),TYPE,Module('ITMacroClass.clw'),Link('ITMacroClass',_ITUtilLi
nkMode_),DLL(_ITUtilDllMode_)

Macros 177                      &IT_Macros
MacroCounter 177                Long
AddMacro 178                    Procedure(String pMacro, String pValue)
ExpandMacro 178                 Procedure(String pMacro),String
ExpandReplace 178               Procedure(String pStr),String  ! Searches and replaces
all macros in string
Construct 178                   Procedure
Destruct 178                    Procedure
                        End
```

3.20.2 Properties Macro Class

```
Macros 177                      &IT_Macros
MacroCounter 177                Long
```

3.20.2.1 MacroCounter Macro Class - Properties

Enter topic text here.

3.20.2.2 Macros Macro Class - Properties

Enter topic text here.

3.20.3 Methods Macro Class

```
AddMacro 178                    Procedure(String pMacro, String pValue)
ExpandMacro 178                 Procedure(String pMacro),String
ExpandReplace 178               Procedure(String pStr),String  ! Searches and replaces
all macros in string
Construct 82                    Procedure
Destruct 82                     Procedure
```

3.20.3.1 Destruct

Enter topic text here.

3.20.3.2 Construct

Enter topic text here.

3.20.3.3 ExpandReplace

Enter topic text here.

3.20.3.4 ExpandMacro

Enter topic text here.

3.20.3.5 AddMacro

Enter topic text here.

3.20.4 Equates

Enter topic text here.

3.21 Network Class

3.21.1 Overview Network Class

```
ITNetworkClass
Class(ITShellClass),TYPE,Module('ITNetworkClass.clw'),Link('ITNetworkClass',_ITUtil
LinkMode_),DLL(_ITUtilDllMode_)

Is98NetCompatible 181          Byte
NetEnumOpen 182                Byte,PRIVATE
NetResources 182               &IT_NETRESOURCEs
LocalResources 182             &IT_NetworkShares
HideDebugView 181              Byte

CheckLeadingBackSlash 183      Procedure(STRING pPath), String
CheckTrailingBackSlash 183     Procedure(STRING pPath), String
EnumLocalShares 185            Procedure(),Long                ! Returnes the number
of enumerated resources
EnumLocalSharesWin32 185       Procedure(),Long,PRIVATE        ! Returnes the number
of enumerated resources
EnumLocalSharesWinNT 185       Procedure(),Long,PRIVATE        ! Returnes the number
of enumerated resources
EnumNetworkDrives 185          Procedure(),LONG,PROC           ! Returns number of
enumerated resources
EnumNetworkPrinters 186        Procedure()
GetLocalNetworkFileName 186    Procedure(String pFile),String  ! Returns the local or
remote filename (UNC)
GetNetworkDriveName 187        Procedure(String pDrive),String ! Returns the remote
drive name
GetNetworkFileName 188         Procedure(String pFile),String  ! Returns the remote
filename (UNC)
IsLocalShare 189               Procedure(String pPath), Byte   ! Returns true if the
path is on a local share
IsUNC 190                      Procedure(String pPath), Byte   ! Returns true if the
path starts with \\
PTD 190                        Procedure(String pS, Byte pHideDebug=False),VIRTUAL
ParseKeyData 190               Procedure(String pData),PRIVATE
ShowLocalShares 191            Procedure,PRIVATE

Construct 191                  Procedure
Destruct 191                   Procedure
                          End
```

3.21.2 Debugging Network Class

The methods in the ITNetwork class have debug code in them that uses OutputDebugString api call to send the information to an external tool. In order to view the debug information you need a tool such as DebugView from www.sysinternals.com. **DebugView is a free tool** and can be used on multiple computers so you can have the debug information show up on a second computer and view it as your program runs on another computer.

In order to turn the debug logging on in your application, simply run your program with /ITNETWORK command line parameter:

MyProgram.exe /ITNetwork

The command line parameter is not case sensitive. /ITNetwork, /ITNETWORK or /itnetwork will all give the same effect.

Below is an example output from a call to GetLocalNetworkFileName 187 on a computer called COMPAQ3200 with a local share of C:\ called PRES_200. Notice that all lines coming from the ITNetworkClass are prefixed with "ITNetwork:" You can use this as a filter in

```
 1   3.12749410   [2908] ITNetwork: Construct, Hide DebugView = False
 2   12.32415581  [2908] ITNetwork: GetLocalNetworkFileName Begins
 3   12.32423476  [2908] ITNetwork: GetNetworkFilename, WNetGetUniversalName = 2250

 4   12.32429489  [2908] ITNetwork: pFile   = C:\Installations\aawsepersonal.exe
 5   12.32433433  [2908] ITNetwork: UncName = C:\Installations\aawsepersonal.exe
 6   12.32437811  [2908] ITNetwork: Filename: C:\Installations\aawsepersonal.exe
 7   12.32441549  [2908] ITNetwork: Need to find local share name
 8   12.32457083  [2908] ITNetwork: Before GetComputerName
 9   12.32460993  [2908] ITNetwork: GetComputerName = ABCOMPAQ3200
10   12.32464809  [2908] ITNetwork: After GetComputerName
11   12.32465735  [2908] ITNetwork: Computer Name: ABCOMPAQ3200
12   12.32479729  [2908] ITNetwork:
13   12.32483673  [2908] ITNetwork:
-----------------------------------------------------
14   12.32490315  [2908] ITNetwork:  Enumerated Local Shares
15   12.32494094  [2908] ITNetwork: CCSFlags:    538976288
16   12.32497734  [2908] ITNetwork: MaxUses:     -1
17   12.32501403  [2908] ITNetwork: Name:        PRES_C200
18   12.32505090  [2908] ITNetwork: Path:        C:\
19   12.32508669  [2908] ITNetwork: Permissions: 0
20   12.32512336  [2908] ITNetwork: Type:        0
21   12.32516002  [2908] ITNetwork: --------------------
22   12.32519856  [2908] ITNetwork:
-----------------------------------------------------
23   12.32524488  [2908] ITNetwork: EnumLocalShares, Before Return, KeyIndex = 1
24   12.32528265  [2908] ITNetwork: After EnumLocalShares
25   12.32532021  [2908] ITNetwork: 1 local shares found
26   12.32535735  [2908] ITNetwork: Path = C:\
27   12.32540102  [2908] ITNetwork: Name = PRES_C200
28   12.32544396  [2908] ITNetwork: Share found: PRES_C200
29   12.32545328  [2908] ITNetwork: Share found:
\\ABCOMPAQ3200\PRES_C200\Installations\aawsepersonal.exe
30   12.32549089  [2908] ITNetwork: UNCName =
\\ABCOMPAQ3200\PRES_C200\Installations\aawsepersonal.exe
31   12.32552804  [2908] ITNetwork: GetLocalNetworkFileName Ends
32   12.44182026  [2908] ITNetwork: Construct, Hide DebugView = False
```

See also:
PTD 190

3.21.3 Data Types — Network Class

The Network Class uses two data types, IT_NETRESOURCEs 181 and IT_NetworkShares 181. Both are queues declared in the ITEquates.inc file.

3.21.3.1 IT_NETRESOURCEs

This queue is used for the NetResources 182 property, but that queue is not used in the class yet. Reserved for future use.

```
IT_NETRESOURCEs          QUEUE,TYPE
Scope                      ULONG
Type                       ULONG
DisplayType                ULONG
Usage                      ULONG
LocalName                  Cstring(256)
RemoteName                 Cstring(256)
Comment                    Cstring(256)
Provider                   Cstring(256)
                         END
```

3.21.3.2 IT_NetworkShares

This queue is used for the LocalResources 182 property which is used by several methods in the Network Class 179.

```
IT_NetworkShares         QUEUE,TYPE
CCSFlags                   LONG
MaxUses                    UNSIGNED
Name                       CString(IT_MAX_PATH)
Path                       CString(IT_MAX_PATH)
Permissions                LONG
Type                       LONG
                         END
```

3.21.4 Properties — Network Class

There are currently 5 properties in the Network Class.

```
HideDebugView 181          Byte
Is98NetCompatible 181      Byte
LocalResources 182         &IT_NetworkShares
NetResources 182           &IT_NETRESOURCEs
NetEnumOpen 182            Byte,PRIVATE
```

3.21.4.1 HideDebugView

The HideDebugView property is used in various methods and is used to determine if debug information should be send to OutputDebugString or not. If the program has /ITNETWORK switch in the command line the debug information will be sent to Debugger by the ODS 179 method.

It is declared as:

```
HideDebugView              Byte
```

3.21.4.2 Is98NetCompatible

This property is true if the Operating System is a Windows 98/95/ME. It is false if the Operating System is Windows NT/2000/XP.

3.21.4.3 LocalResources

Queue of IT_NetworkShares 181 type.

```
IT_NetworkShares          QUEUE,TYPE
CCSFlags                    LONG
MaxUses                     UNSIGNED
Name                        CString(IT_MAX_PATH)
Path                        CString(IT_MAX_PATH)
Permissions                 LONG
Type                        LONG
                          END
```

3.21.4.4 NetEnumOpen

Private property. Used internally to determine if the network enumeration process was opened successfully or not.

3.21.4.5 NetResources

Queue of IT_NETRESOURCEs 181 type.

```
IT_NETRESOURCEs           QUEUE,TYPE
Scope                       ULONG
Type                        ULONG
DisplayType                 ULONG
Usage                       ULONG
LocalName                   Cstring(256)
RemoteName                  Cstring(256)
Comment                     Cstring(256)
Provider                    Cstring(256)
                          END
```

3.21.5 Methods

There are currently 17 methods in the Network Class.

```
CheckLeadingBackSlash 183     Procedure(STRING pPath), String
CheckTrailingBackSlash 183    Procedure(STRING pPath), String
EnumLocalShares 185           Procedure(),Long                ! Returnes the number
of enumerated resources
EnumLocalSharesWin32 185      Procedure(),Long,PRIVATE        ! Returnes the number
of enumerated resources
EnumLocalSharesWinNT 185      Procedure(),Long,PRIVATE        ! Returnes the number
of enumerated resources
EnumNetworkDrives 185         Procedure(),LONG,PROC           ! Returns number of
enumerated resources
```

```
EnumNetworkPrinters 186          Procedure()
GetLocalNetworkFileName 186      Procedure(String pFile),String  ! Returns the local or
remote filename (UNC)
GetNetworkDriveName 187          Procedure(String pDrive),String ! Returns the remote
drive name
GetNetworkFileName 188           Procedure(String pFile),String  ! Returns the remote
filename (UNC)
IsLocalShare 189                 Procedure(String pPath), Byte   ! Returns true if the
path is on a local share
IsUNC 190                        Procedure(String pPath), Byte   ! Returns true if the
path starts with \\
PTD 190                          Procedure(String pS, Byte pHideDebug=False),VIRTUAL
ParseKeyData 190                 Procedure(String pData),PRIVATE
ShowLocalShares 191              Procedure,PRIVATE

Construct 191                    Procedure
Destruct 191                     Procedure
```

3.21.5.1 CheckLeadingBackSlash

Prototype: **(STRING pPath), String**

Takes a path as parameter. Checks if the path begins with backslash and if it does not, it adds a leading backslash.

Example:

```
MyProc   Proc
P    String(255)
ITN  ITNetworkClass
 Code
 P = 'somepath\myfile.txt'
 P = ITN.CheckLeadingBackSlash(P)
 Message('P is now = ' & P)
```

P would now be '\somepath\myfile.txt'

This is mostly for internal use in the class, but is made public in case someone needs it.

See also:
CheckTrailingBackSlash 183

3.21.5.2 CheckTrailingBackSlash

Prototype: **(STRING pPath), String**

Takes a path as parameter. Checks if the path ends with backslash and if it does not, it adds a leading backslash.

Example:

```
MyProc   Proc
P    String(255)
ITN  ITNetworkClass
 Code
 P = 'somepath\myfile.txt'
```

```
P = ITN.CheckLeadingBackSlash(P)
Message('P is now = ' & P)
```

P would now be '\somepath\myfile.txt'

This is mostly for internal use in the class, but is made public in case someone needs it.

See also:
CheckLeadingBackSlash 183

3.21.5.3 ConvertToUNC

Prototype: **(String pFile),String**

pFile Filename of a local or shared/mapped file

Returns UNC for a local or shared file

This method takes a filename that includes a drive letter, local or mapped, such as "F:\Clarion6\Bin\C60HELP.HLP" and returns the correctly formed UNC name for the file, i.e. "\\ComputerName\DriveShareName\Clarion6\Bin\C60HELP.HLP"

Note that this method will convert any filename to a UNC filename, also filenames on the local computer. This method calls GetLocalNetworkFileName 186

Example:

```
ITN  ITNetworkClass
Fn   String(1024)
UNC  String(1024)
 Code
 If FileDialog('Choose File to View',Fn,'Text|*.TXT|Source|*.CLW',FILE:LongName)
   UNC = ITN.ConvertToUNC(Fn)
   Message('File Name: ' & Fn & '|UNC Name:  ' & UNC)
 End
```

See also:
GetUNCFileName 188
GetNetworkFileName 188
GetLocalNetworkFileName 186
EnumLocalShares 185
ComputerName 54
LocalResources 182

3.21.5.4 ConvertToAscii

Not implemented

3.21.5.5 EnumLocalShares

Network Class - Methods

Prototype: **()**

Primary method to enumerate local shares. This method checks for the Computer name 63 and if no computer name is detected, it indicates that there are no local shares so it skips the process. Since the storage information for local shares is different on 95/98/ME than it is on NT/2000/XP, this method determines which operating system is in use and then calls the EnumLocalSharesWin32 185 or EnumLocalSharesWinNT 185 depending on the result of the call to GetWindowVersion()

For example code, see the source for the GetLocalNetworkFileName 187 method.

See also:
EnumLocalSharesWin32 185
EnumLocalSharesWinNT 185

3.21.5.6 EnumLocalSharesWin32

Network Class - Methods

Prototype: **(), Long**

Returns the number of enumerated local shares. This method enumerates information that is stored in the "LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\Lanman" registry key.

See also:
EnumLocalShares 185
EnumLocalSharesWinNT 185

3.21.5.7 EnumLocalSharesWinNT

Network Class - Methods

Prototype: **(), Long**

Returns the number of enumerated local shares. This method enumerates information that is stored in the "LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\lanmanserver\Shares" registry key.

See also:
EnumLocalShares 185
EnumLocalSharesWin32 185

3.21.5.8 EnumNetworkDrives

Network Class - Methods

Prototype: **(), LONG, PROC**

This method is not functional yet.

This method enumerates remote shared network resources, i.e. it does not enumerate local shared

resources. This can be used to find all available network resources.

3.21.5.9 EnumNetworkPrinters

Prototype: **()**

This method is not yet completed (as of November 30, 2010)

Example:

See also:

3.21.5.10 GetLocalNetworkFileName

Prototype: **(String pFile),String**

pFile Filename of a local or shared/mapped file

Returns UNC for a local or shared file

This method takes a filename that includes a drive letter, local or mapped, such as "F:\Clarion6\Bin\C60HELP.HLP" and returns the correctly formed UNC name for the file, i.e. "\\ComputerName\DriveShareName\Clarion6\Bin\C60HELP.HLP"

Note that this method will convert any filename to a UNC filename, also filenames on the local computer. This method has two other names, ConvertToUNC 184 and GetUNCFileName 188. They both call GetLocalNetworkFileName

Example:

```
ITN  ITNetworkClass
Fn   String(1024)
UNC  String(1024)
 Code
 If FileDialog('Choose File to View',Fn,'Text|*.TXT|Source|*.CLW',FILE:LongName)
   UNC = ITN.GetLocalNetworkFileName(Fn)
   Message('File Name: ' & Fn & '|UNC Name:  ' & UNC)
 End
```

See also:

ConvertToUNC 184

GetUNCFileName 188

GetNetworkFileName 188

EnumLocalShares 185

ComputerName 54

LocalResources 182

3.21.5.11 GetLocalNetworkFileName

Network Class - Methods

Prototype: **(String pFile),String**

This method takes a filename as parameter and will return the appropriate UNC filename for it, no matter if it is a local file or a remote file (on another computer), using the enumeration functions to find the appropriate name. Note that if no UNC name can be established it will return the original name back, so this method should be 100% safe on standalone computers where there are no local or remote shares to connect to. In that case it will simply return the filename/path passed to it.

Example:

```
MyProc   Proc
P    String(255)
ITN  ITNetworkClass
 Code
 P = 'c:\somepath\myfile.txt'
 P = ITN.GetLocalNetworkFileName(P)
 Message('P is now = ' & P)
```

P would now be '\\COMPAQ\TheCDrive\somepath\myfile.txt' if the computer name was "COMPAQ" and the local share name for the C:\ was "TheCDrive". If there was no local share, this would return "c:\somepath\myfile.txt" even if the computer name was still "COMPAQ" If you want to provide an option later on to convert to UNC, it would be advisable to store the computer name with the filename, that way the filename(s) could be fairly easily converted to UNC.

See also:
CheckLeadingBackSlash 183
EnumLocalShares 185
GetNetworkFileName 188
LocalResources 182

3.21.5.12 GetNetworkDriveName

Network Class - Methods

Prototype: **(String pDrive), String**

This method returns the UNC drive name for a mapped drive.

Example:

```
MyProc   Proc
P    String(255)
ITN  ITNetworkClass
 Code
 P = 'Z:'
 P = ITN.GetNetworkDriveName(P)
 Message('P is now = ' & P)
```

If the Z: drive is mapped to "RemoteC" drive share on "RemotePC" computer, this would return

"\\RemotePC\RemoteC"

See also:
GetComputerName 63
GetLocalNetworkFileName 187
GetNetworkFileName 188

3.21.5.13 GetNetworkFileName

Network Class - Methods

Prototype: **(String pFile), String**

This method returns the UNC name for a remote file. It does NOT return the UNC filename for a local file. Use GetLocalNetworkFileName 187 for that purpose.

Example:

```
MyProc   Proc
P    String(255)
ITN  ITNetworkClass
 Code
 P = 'Z:\somepath\somefile.txt'
 P = ITN.GetNetworkFileName(P)
 Message('P is now = ' & P)
```

If the Z: drive is mapped to "RemoteC" drive share on "RemotePC" computer, this would return "\\RemotePC\RemoteC\somepath\somefile.txt" If the Z: drive is a local drive, the GetNetworkFileName 188 would return "Z:\somepath\somefile.txt"

See also:
GetComputerName 63
GetLocalNetworkFileName 187
GetNetworkDriveName 187

3.21.5.14 GetUNCFileName

Network Class - Methods

Prototype: **(String pFile),String**

pFile Filename of a local or shared/mapped file

Returns UNC for a local or shared file

This method takes a filename that includes a drive letter, local or mapped, such as "F:\Clarion6\Bin\C60HELP.HLP" and returns the correctly formed UNC name for the file, i.e. "\\ComputerName\DriveShareName\Clarion6\Bin\C60HELP.HLP"

Note that this method will convert any filename to a UNC filename, also filenames on the local computer. This method calls GetLocalNetworkFileName 186

Example:

```
ITN  ITNetworkClass
Fn   String(1024)
```

```
UNC  String(1024)
 Code
 If FileDialog('Choose File to View',Fn,'Text|*.TXT|Source|*.CLW',FILE:LongName)
   UNC = ITN.GetUNCFileName(Fn)
   Message('File Name: ' & Fn & '|UNC Name:  ' & UNC)
 End
```

See also:
ConvertToUNC 184

GetNetworkFileName 188

GetLocalNetworkFileName 186

EnumLocalShares 185

ComputerName 54

LocalResources 182

3.21.5.15 IsLocalShare

Network Class - Methods

Prototype: **(String pPath), Byte**

pPath UNC path

Returns Returns True if the path is part of a local share, false if it is not.

This method first checks if the pPath is a UNC. This is done by simply checking if the first two characters are '\\' Then local shares are enumerated and each one is searched for the pPath. If a local share is found that matches part of the pPath UNC path, then it is considered to be a match. pPath can include a filename, for example "\\comp\C_Drive\Clarion\Apps\MyProgram\MyProgram.app". If a local share is found that contains "\\comp\C_Drive\Clarion" then IsLocalShare will return True.

Note that it does not care how many local shares may be pointing to the path, for example there could be one pointing to "\\comp\C_Drive", one to "\\comp\C_Drive\Clarion" and one to "\\comp\C_Drive\Clarion\Apps" and IsLocalShare will simply determine up on first find that the path is part of a local share and return true.

Example:

```
ITN  ITNetworkClass
 Code
 If ITN.IsLocalShare('\\comp\C_drive\Clarion\Apps\MyProgram\MyProgram.app')
   Message('This file is on a local share')
 End
```

See also:
ConvertToUNC 184

GetLocalNetworkFileName 186

GetUNCFileName 188

EnumLocalShares 185

3.21.5.16 IsUNC

Prototype: **(String pPath), Byte**

pPath Path to check if it is a UNC path or not

Returns Returns true or false depending on if the pPath path starts with '\\' or not

This method is very simple as it just checks if the path is more than two characters and if those two characters are '\\' If that is the case the method will return true, otherwise it will return false.

Example:

```
ITN  ITNetworkClass
P  String(255)
 Code
 P = 'C:\'
 If ITN.IsUNC(P)
   Message(Clip(P) & ' is a UNC path')
 Else
   Message(Clip(P) & ' is NOT a UNC path')
 End
```

See also:

ConvertToUNC 184

IsLocalShare 189

GetLocalNetworkFileName 186

GetUNCFileName 188

3.21.5.17 ParseKeyData

Prototype: **(String pData)**

This method is a private method that is used to parse data for the LocalResource queue.

3.21.5.18 PTD

Prototype: **(String pS, Byte pHideDebug=False),VIRTUAL**

This method is a virtual method that uses the PTD in the parent class to PrintToDebug, i.e. call the OutputDebugString api call to send debugging information to tools such as DebugView.

See also:

Debugging 179

3.21.5.19 ShowLocalShares

Prototype: **(none)**

This method sends all records in the LocalResources 182 queue to DebugView. It is convenient for debugging purposes only.

Example:

```
ITN  ITNetworkClass
 Code
 Message('Run DebugView')
 ITN.EnumLocalShares
 ITN.ShowLocalShares
 Message('Check DebugView for list of local shares')
```

See also:
EnumLocalShares 185

3.21.5.20 Construct

The constructor initializes the NetResources 182 and LocalResources 182 queues. It also gets the window version and sets the Is98NetCompatible 181 property to true if the VersionPlatformID is set to IT_VER_PLATFORM_WIN32_WINDOWS.

3.21.5.21 Destruct

The Destructor frees the NetResource 182 and LocalResoucers 182 and disposes of them.

3.22 Page Of Pages Class

3.22.1 Overview Page Of Pages Class

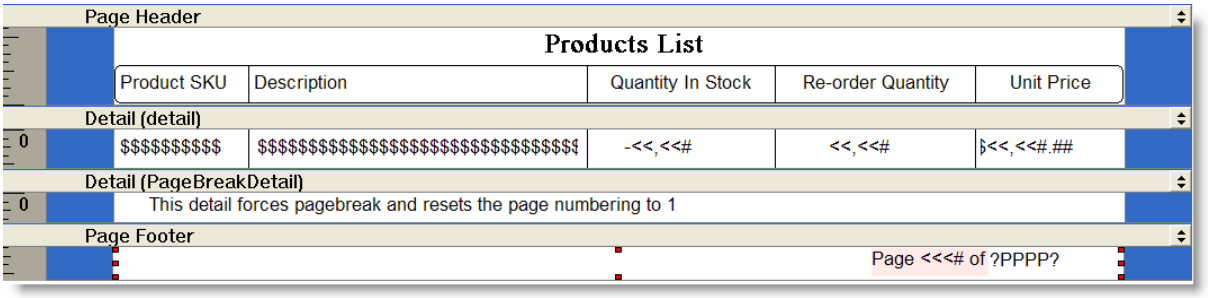
The Page of Pages class can be used with any kind of Clarion report. It can be implemented in hand coded reports as well as in reports in applications. Both Icetips Previewer and Icetips Utilities includes a template that makes it easy to implement on reports.

As of June 2012, the Icetips Utilities also include a template 426 to implement the Page of Pages class on your reports. Also if you are using the Icetips Previewer we suggest that you use that template as it is specifically designed for the Previewer.

Please note that this Page of Pages template is NOT compatible with the Legacy/Clarion template chain when using the Icetips Previewer. For that use the Page of Pages template in the Icetips Previewer. Apart from that difference the two templates are practically identical.

In normal operation, you only need to call two methods, the Init 197 method and the SetPageOfPages 198. The rest can really be treated as private or protected methods as they are only used internally.

To implement, first add the Icetips Global Utilities template to the application. The next thing is to add the page number on the report as in:



The "Page <<<# of" control is a page number control defined with a picture of "@pPage <<<# of p" and shows up in the report structure as:

```
STRING(@pPage <<<# of p),AT(5625,8,865,208),PAGENO,USE(?PageCount),FONT('Arial',10,,),
```

The ?PPPP? control immediately to the right of the page number control is a normal string control that is used by the Page of Pages class. This control will get the total number of pages to place into the control. If it is dropped on by the Previewer control template, then it will be created like this in the report structure:

```
STRING('?PPPP?'),AT(6500,17),USE(?ITPPageOfPages),LEFT,#SEQ(2),#ORIG(?ITPPageOfPages)
```

On the report procedure you need to add a few lines of code.

1. In the Local data embed add:

```
ITPOP  ITPageOfPagesClass
```

2. In the ThisWindow.OpenReport embed, after Parent Call, put:

```
ITPOP.Init(Report,SELF.PreviewQueue,'?PPPP?')
```

The first parameter is the report structure label, then it is the image queue used for the metafile pages. The '?PPPP?' string in the final parameter MUST match the string on the report, i.e. the

STRING('?PPPP?')

3. Immediately before the report is previewed, such as the ThisWindow.AskPreview embed, add:

```
ITPOP.SetPageOfPages
```

That's it! If you are using the Icetips Previewer all you need to do is drop the control template on the report and you are done.

If you want to reset the total page number, for example if you are printing invoices that can multiple pages, you can simply call the SetPageOfPages method after printing each detail. The class keeps track of which imagefiles it has updated and will not check any files that have already been updated. The SetPageOfPages is very fast and you will not see much impact from this on your report performance. Here is an example of code put into ThisWindow.TakeRecord after Parent call, to split a report up based on the first letter in the product name, if the name starts with the letter 'G' or later:

```
If PRO:ProductSKU[1] > 'F'
   If PRO:ProductSKU[1] <> Clip(Loc:LastProduct)
     Print(RPT:PageBreakDetail)
     ITPOP.SetPageOfPages
     Loc:LastProduct = PRO:ProductSKU[1]
   End
 End
```

Is it possible that this will fail if the metafile happens to have the same string in it as you use in your INIT method? Yes, it is possible. However with a string like ?PPPP? it is very unlikely that you will ever run into problems with it. To prevent possible problems with changing the token, it is only replaced if it is only found once and only once in the metafile. So if the search string combination is found twice, then the search token will not be update with the total page count.

3.22.2 Properties — Page Of Pages Class

The PageOfPages class has 8 public properties and one private property.

```
ReportPrevieweQueue 193      &PrintPreviewFileQueue
TotalPages 194               Long
ThisReport 194               &REPORT
PageBuffer 194               &String
SearchString 194             CString(101),Private
PageOf 195                   Long
StartPointer 195             Long
LastPointer 195              Long
TimeTaken 195                Long
```

3.22.2.1 ReportPrevieweQueue — Page Of Pages Class - Properties

The ReportPrevieweQueue property is used in the Init 197 and SetPageOfPages 198 methods and is used to store a reference to the image queue from the report, which contains the filenames of the windows metafiles (.wmf files) that are included in the report.

It is declared as:

```
ReportPrevieweQueue        &PrintPreviewFileQueue
```

The PrintPreviewFileQueue is defined in the EQUATES.CLW Clarion standard file. It is defined like

this:

```
PrintPreviewFileQueue   QUEUE,TYPE
Filename                   STRING(FILE:MaxFileName)
PrintPreviewImage          STRING(FILE:MaxFileName),OVER(Filename)
                        END
```

3.22.2.2 TotalPages

The TotalPages property is used in the SetPageOfPages 198 method to store the total number or pages or records in the ReportPreviewQueue. It is also used in calculations of the PageOf 195 and LastPointer 195 properties.

It is declared as:

```
TotalPages              Long
```

3.22.2.3 ThisReport

The ThisReport property is used in the Init 197 method to store a reference to the report structure that is being printed. Currently it is not used anywhere else in the classes but this property is reserved for future use.

It is declared as:

```
ThisReport              &REPORT
```

3.22.2.4 PageBuffer

The PageBuffer property is used in the ReadTheFile 198, WriteTheFile 200, SetPageOfText 199, AllocatePageBuffer 196 and DisposePageBuffer 196 methods and is used to store the byte stream from the windows metafile (.WMF) that contains the page image. The ReadTheFile 198 and WriteTheFile 200 methods read and write the file into a single buffer, making the operation very fast and efficient.

It is declared as:

```
PageBuffer              &String
```

3.22.2.5 SearchString

The SearchString property is used in the SetPageOfText 199, SetSearchString 199 and GetSearchString 196 methods and is used to store the string being searched for in the windows metafile buffer which is read by the ReadTheFile 198 method. Note that this is a private property and can only be accessed via the SetSearchString 199 and GetSearchString 196.

It is declared as:

```
SearchString            CString(101),Private
```

3.22.2.6 PageOf

The PageOf property is used in the SetPageOfPages 198 and SetPageOfText 199 methods and is used to keep the current page number. This is the X value in the "page X of N" string. When the SetPageOfPages 198 is called the TotalPages 194 and LastPointer 195 properties are set so that the next time SetPageOfPages 198 is called, PageOf restarts the counting. This enables you to use the Page of Pages class for example to put correct page numbers on invoices where a single customer can have one or more pages, even if you are printing out multiple invoices. The "page X of" will be reset for each customer.

It is declared as:

```
PageOf                  Long
```

3.22.2.7 StartPointer

The StartPointer property is used in the SetPageOfPages 198 method and is used to keep track of the first page in the page group that is being updated.

It is declared as:

```
StartPointer            Long
```

3.22.2.8 LastPointer

The LastPointer property is used in the SetPageOfPages 198 and SetPageOfText 199 methods and is used to keep track of the last page that was processed in the group of pages being processed. This enabled the Page of Pages class to break up reports, see the PageOf property 195 for more detail.

It is declared as:

```
LastPointer             Long
```

3.22.2.9 TimeTaken

The TimeTaken property is used in the SetPageOfPages 198 and ProfileToODS 197 methods and is used to keep track of the time it takes to update the report. This property doesn't do anything in the class other than just add up the time. If you are curious, you can check this property at the end of the report after the final call to SetPageOfPages 198 and see how long the entire Page of Pages process took.

It is declared as:

```
TimeTaken               Long
```

3.22.3 Methods

The Page of Pages class has 13 methods including constructor and destructor. Normally you will only ever need to use the Init 197 and SetPageOfPages 198 methods, but I have not marked the others as private or protected as there can be legitimate reasons to use them. See example code in the Overview chapter 192 for more information.

```
AllocatePageBuffer 196          Procedure(Long pBytesToAllocate)
DisposePageBuffer 196           Procedure
GetSearchString 196             Procedure(),String
Init 197                        Procedure(REPORT pReport, PrintPreviewFileQueue pQ, <String
pSearchString>)
ProfileToODS 197                Procedure
ReadTheFile 198                 Procedure(String pFileName, Long pFileSize),Long,PROC ! Returns
bytes read
SetPageOfPages 198              Procedure
SetPageofText 199               Procedure(),Byte ! Returns false if nothing was replaced, true
if needs to be written back
SetSearchString 199             Procedure(String pSearchString)
UpdatePageFile 200              Procedure(String pFile, Long pFileSize),Byte
WriteTheFile 200                Procedure(String pFileName, Long pFileSize),Long,PROC ! Returns
bytes written
Construct 200                   Procedure
Destruct 201                    Procedure
```

3.22.3.1 AllocatePageBuffer

Prototype: **(Long pBytesToAllocate)**

pBytesToAllocate Number of bytes to allocate. The size is equal to the size of the page file (.WMF) being read into the buffer.

This method allocates a string buffer for the entire page file (windows metafile) The pagebuffer is then processed and the search string replaced with the appropriate number for the pages. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
PageBuffer 194

DisposePageBuffer 196

3.22.3.2 DisposePageBuffer

Prototype: **None**

This method disposes of the page buffer string that has been previously allocated by AllocatePageBuffer 196. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
PageBuffer 194

AllocatePageBuffer 196

3.22.3.3 GetSearchString

Prototype: **(),String**

Returns Returns the value of the SearchString 194 property.

This method returns the value/content of the SearchString property 194 which is used to store the string to search for in the metafile.

Example:

```
ITPoP  ITPageOfPagesClass
    CODE
    ITPoP.Init(Report,SELF.PreviewQueue,'?PPPP?')        ! Initialize using
'?PPPP?'
    ITPoP.SetSearchString('?XXXX?')                      ! Change the search string
to '?XXXX?'
    Message('Search String: ' & ITPop.GetSearchString()) ! Search String would
return '?XXXX?'
```

See also:
SearchString 194

SetSearchString 199

3.22.3.4 Init — Page Of Pages Class - Methods

Prototype: **(REPORT pReport, PrintPreviewFileQueue pQ, <String pSearchString>)**

pReport	Reference to the report structure/label
pQ	Reference to the image queue used by the report
pSearchString	String to search for in the report metafile. This must be a unique string that is unlikely ever to be found in a metafile.

This method initializes the Page of Pages class and once this method has been called the SetPageOfPages 198 can be called. This method should be called after the report has been opened and before the preview is called. The method takes the label of the report as the first parameter, then the image queue being used by the report engine. The last parameter is a string that is later replaced with the total page number in the SetPageOfPages 198 method.

Example:

```
ITPoP  ITPageOfPagesClass
    CODE
    ITPoP.Init(Report,SELF.PreviewQueue,'?PPPP?')  ! Initialize using '?PPPP?'
```

See also:
SetPageOfPages 198

SetSearchString 199

3.22.3.5 ProfileToODS — Page Of Pages Class - Methods

Prototype: **None**

This method is for debugging purposes only. It sends information about the time taken by the calls to

SetPageOfPages 198 process, detailing how long the search/replace operation took. Unless you experience performance problems, you never need to call this method.

Example:

```
ITPoP  ITPageOfPagesClass
    CODE
    ITPoP.Init(Report,SELF.PreviewQueue,'?PPPP?')
    !! Print ton of pages here...
    ITPoP.SetPageOfPages  ! Process the metafiles.
    ITPoP.ProfileToODS    ! Send the timing information to DebugView
```

See also:
TimeTaken 195
SetPageOfPages 198

3.22.3.6 ReadTheFile

Prototype: **(String pFileName, Long pFileSize),Long,PROC ! Returns bytes read**

pFileName Name of the windows metafile to read.
pFileSize The size of the file

Returns Returns the number of bytes read. That should always match the value in pFileSize.

This method allocates buffer for the windows metafile and reads it into the PageBuffer 194 property. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
PageBuffer 194
AllocatePageBuffer 196
DisposePageBuffer 196

3.22.3.7 SetPageOfPages

Prototype: **None**

This is the work horse method of the class. You call this method every time that you want the metafiles to be update. Note that when you call this method the total page counter is reset so the next page will be the first in a new total.

For example if you are printing 3 invoices. The first one is 2 pages the second is 1 page and the last one is 4 pages. On the first invoice the pages would be "page 1 of 2" and "page 2 of 2" On the second one it would be "page 1 of 1" The last invoice would have "page 1 of 4", "page 2 of 4", "page 3 of 4" and "page 4 of 4" To accomplish this you simply call SetPageOfPages 198 after the pagebreak for the last page of the invoice has happened.

Note that if you call the method multiple times while the same page is printing, it will not attempt to update anything that has been updated already. This saves a lot of time since the class keeps track of the pages that have been updated with the correct page numbering and doesn't touch them again.

Example:

```
! Print pages with different total pages.  Separated by the product SKU.
If PRO:ProductSKU[1] > 'F'
   If PRO:ProductSKU[1] <> Clip(Loc:LastProduct)
     Print(RPT:PageBreakDetail)
     ITPOP.SetPageOfPages   ! Call SetPageOfPages after each Print statement to ensure
it updates the pages
     Loc:LastProduct = PRO:ProductSKU[1]
   End
 End
```

See also:
Init 197
GetSearchString 196
SetSearchString 199

3.22.3.8 SetPageofText — Page Of Pages Class - Methods

Prototype: **(),Byte**

Returns Returns true if the string was replaced, false if it was not replaced. True indicates that the metafile needs to be written.

This method is responsible for actually replacing the SearchString 194 text in the PageBuffer 194 with the correct page number. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
SetPageOfPages 198
UpdatePageFile 200

3.22.3.9 SetSearchString — Page Of Pages Class - Methods

Prototype: **(String pSearchString)**

pSearchString String to assign to the SearchString 194 property. This string is used by SetPageOfPages 198 as the string to search for in the metafiles.

This method simply sets the SearchString 194 property to the value of the pSearchString parameter.

Example:

```
ITPoP  ITPageOfPagesClass
    CODE
    ITPoP.Init(Report,SELF.PreviewQueue,'?PPPP?')        ! Initialize using
'?PPPP?'
```

```
    ITPoP.SetSearchString('?XXXX?')                    ! Change the search string
to '?XXXX?'
    Message('Search String: ' & ITPop.GetSearchString()) ! Search String would
return '?XXXX?'
```

See also:
SearchString 194
GetSearchString 196

3.22.3.10 UpdatePageFile

Prototype: **(String pFile, Long pFileSize),Byte**

pFile Name of the windows metafile to update.
pFileSize The size of the file

Returns Returns true

This method does the reading, updating and writing of a single file/page. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
ReadTheFile 198
SetPageOfPages 198
WriteTheFile 200

3.22.3.11 WriteTheFile

Prototype: **(String pFileName, Long pFileSize),Long,PROC**

pFileName Name of the windows metafile to write.
pFileSize The size of the file

Returns Returns the number of bytes written to the metafile. This should always match the value of pFileSize. The method can be called as a procedure.

This method is responsible for writing the PageBuffer 194 back to the windows metafile after the SearchString 194 has been replaced with the correct page number values. This method should not be called outside of the Page of Pages class and should be considered private method.

See also:
PageBuffer 194
SearchString 194
ReadTheFile 198
SetPageOfPages 198

3.22.3.12 Construct

Prototype: **None**

The Construct method is currently empty.

See also:
Destruct 201

3.22.3.13 Destruct

Prototype: **None**

The Destructor method calls the DisposePageBuffer 196 method to dispose of any buffer that may be left allocated.

See also:
PageBuffer 194
DisposePageBuffer 196
Construct 200

3.23 Period Class

3.23.1 Overview Period Class

```
ITPeriodClass
CLASS(ITStringClass),TYPE,Module('ITPeriodClass.clw'),Link('ITPeriodClass',_ITUtilL
inkMode_),DLL(_ITUtilDllMode_)

Years                       &ITPerQ
Months                      &ITPerQ
Days                        &ITPerQ
YearsFEQ                    Long
MonthsFEQ                   Long
DaysFEQ                     Long
SelectedYear                Long
SelectedMonth               Long
SelectedDay                 Long
SelectedDate                Long
SelectedDateFEQ             Long
FirstYear                   Long
LastYear                    Long
YearVar                     ANY
MonthVar                    ANY
DayVar                      ANY
LastFEQ                     Long,Private
MonthNames                  CString(1025)
PeriodType                  Byte,Private

Init                        Procedure(Long pYearList, Long pMonthList, Long
pDaysList, <Long pSelectedDate>, *? pYear, *? pMonth, *? pDay)
Kill                        Procedure
LoadQueues                  Procedure
LoadYears                   Procedure
LoadMonths                  Procedure
LoadDays                    Procedure(Long pYear=0, Long pMonth=0)
AddToQueue                  Procedure(ITPerQ pQ, String pStrVal, Long
pValue),Private
AddYearRange                Procedure(Long pFirstYear, Long pLastYear)
SetMonthNames               Procedure(<String pMonthNames>)
AdjustDropDowns             Procedure
GetMonthDays                Procedure(Long pYear, Long pMonth),Byte
GetCalculatedDate           Procedure(Long pYear=0, Long pMonth=0, Long pDay=0),Long
GetCalculatedDateStr        Procedure(Long pYear=0, Long pMonth=0, Long
pDay=0),String
GetSelectedDate             Procedure(),Long
TakeEvent                   Procedure,Byte
ResetDropDowns              Procedure
Construct                   Procedure
Destruct                    Procedure
                          End
```

3.23.2 Properties Period Class

Enter topic text here.

3.23.3 Methods Period Class

Enter topic text here.

3.24 Popup Class

3.24.1 Overview Popup Class

Enter topic text here.

3.24.2 Properties Popup Class

Enter topic text here.

3.24.3 Methods Popup Class

Enter topic text here.

3.25 Progress Class

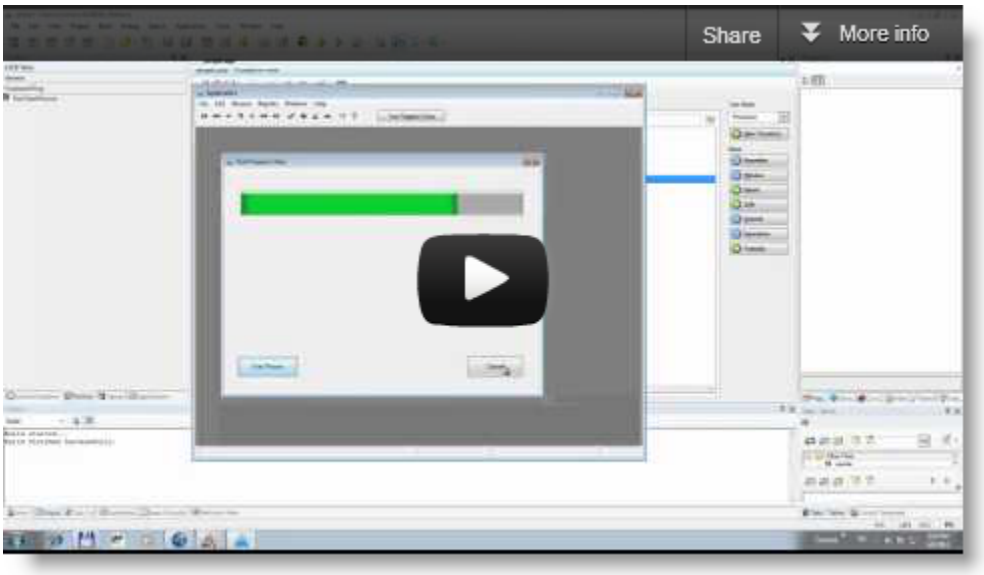
3.25.1 Overview Progress Class

The Progress Class is designed to handle progress bars. In its simplest form it can be used with 3 statements, Init, Update and Kill. Before a loop the Init is called to initialize what progress control to use and how many records will be processed. Update is called during the loop to update the progress bar and after the loop Kill is called to hide the progress bar.

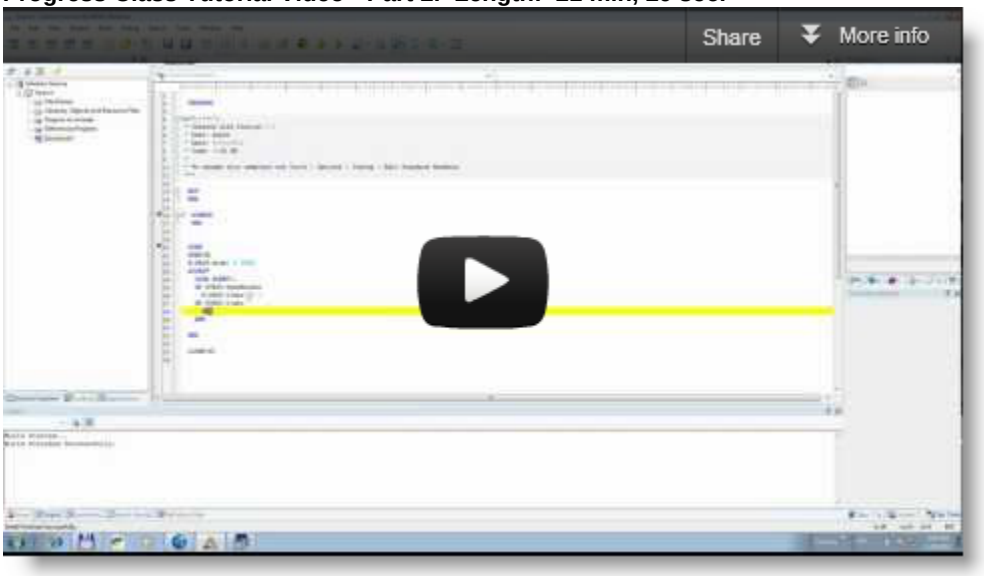
In Windows Vista™ Microsoft changed the way progress bars work, where the progress bar is being updated by a separate thread. This could lead to progress bars being completely off, either completing long before the process was done, or never go all the way even if the process was done. The Icetips Progress class uses a simple trick to make the progress bar stay right on target no matter what operating system your application is running on!

```
ITProgressClass
Class(ITUtilityClass),TYPE,Module('ITProgressClass.clw'),Link('ITProgressClass',_IT
UtilLinkMode_),DLL(_ITUtilDllMode_)
CurrentValue 208              Long,Private
DisplayControls 208           &ITDisplayQueue
HideUnhide 209                Byte
Initialized 208               Byte
PercentValue 209              Byte,Private
ProgressControl 209           Long,Private
TotalValue 209                Long,Private

AddDisplayControl 210         Procedure(LONG pControLToDisplay,Byte pUpdateOnShow=1)
AddToCurrentValue 211         Procedure(Long pValueToAdd),LONG,PROC ! Adds value and
returns new current value
Calculate 211                 Procedure                     ! Calculates percent
GetCurrentPercent 212         Procedure(),BYTE              ! Returns 0 - 100
GetCurrentValue 212           Procedure(),LONG              ! Returns CurrentValue
GetProgressControl 213        Procedure(),LONG              ! Returns
ProgressControl
GetTotalValue 213             Procedure(),LONG              ! Returns TotalValue
HideControls 214              Procedure(BYTE pHide)         ! Hide/unhide display
controls
Init 214                      Procedure(LONG pProgressControl, Long pTotalValue,
Byte pHideUnhide=1, Byte pCanBeZeroOrOne=True)
Kill 215                      Procedure
ReleaseWindow 216             Procedure
SetCurrentValue 217           Procedure(Long pCurrentValue)
SetTotalValue 218             Procedure(LONG pTotalValue)
ShowProgress 218              Procedure
ShowUpdateProgress 219        Procedure(Long pValueToAdd)   ! Update progressbar
after adding pValueToAdd to the value.
Update 219                    Procedure ! Calls ShowUpdateProgress(1)
                          End
```

Progress Class Tutorial Video - Part 1. Length: 15 min, 13 sec.



Progress Class Tutorial Video - Part 2. Length: 21 min, 29 sec.



Example 1 - loop through a queue in a tight loop:

```
ITP  ITProgressClass
I      Long
Q    Queue
F1     String(100)
     End
 Code
 Do FillQueue
 ITP.Init(?Progress1,Records(Q))
 Loop I = 1 To Records(Q)
   Get(Q,I)
   ITP.Update
 End
 ITP.Kill
```

Example 2 - loop through a file in a timer loop:

```
Window WINDOW('Please wait...'),AT(,,159,44),GRAY
       PROGRESS,USE(?Progress1),AT(4,6,151,15),RANGE(0,100)
       BUTTON('Cancel'),AT(57,26,45,14),USE(?CancelButton),HIDE
     END
ITP  ITProgressClass
RecsPrTimer Equate(100)
Done        Byte
 Code
 Open(Window)
 Open(MyFile)
 Set(MYF:MyKey)
 ITP.Init(?Progress1,Records(MYF:MyKey),True) !! Initialize and tell it to handle more controls...
 ITP.AddDisplayControl(?CancelButton,False)   !! ... and add the ?CancelButton...
 ITP.HideControls(False)                      !! ... and unhide all handled controls while the progress is
 Accept
 Case Event()
 Of EVENT:OpenWindow
   Window{Prop:Timer} = 1
 Of EVENT:Timer
   If Done
     Window{Prop:Timer} = 0
     Break !! Break Accept loop
   End
   Loop RecsPrTimer Times
     Next(MyFile)
     If ErrorCode()
       Done = True
       Break
     End
     !! Do something with the MYF: Record
     ITP.Update                                 !! Update the progress bar
   End
   Case Field()
   Of ?CancelButton
     Case Event()
     Of EVENT:Accepted
       Done = True
       Window{Prop:Timer} = 0
       Break !! Break Accept Loop
     End
   End
 End
 ITP.Kill                                       !! Terminate class, hide progress bar and any additional cont
 Close(MyFile)
 Close(Window)
```

3.25.2 Data Types — Progress Class

The Progress class has one Data Type:

```
ITDisplayQueue 207          QUEUE,TYPE
ITDisplayControl              Long
UpdateOnShow                  Byte  !Should the control be displayed on each call to ShowProgress?
                            END
```

3.25.2.1 ITDisplayQueue — Progress Class - Data Types

Queue to store controls that should be made visible and hidden along with the progress bar. This can come in handy for percent string or strings for other indicators of how the progress is going. See the DisplayControls 208 property and AddDisplayControl 210 method for more information.

```
ITDisplayQueue            QUEUE,TYPE
ITDisplayControl            Long
UpdateOnShow                Byte  !Should the control be displayed on each call to
ShowProgress?
```

```
END
```

3.25.3 Properties — Progress Class

There are currently 7 properties in the Progress class:

CurrentValue 208	Long,Private
DisplayControls 208	&ITDisplayQueue 207
HideUnhide 209	Byte
Initialized 208	Byte
PercentValue 209	Byte,Private
ProgressControl 209	Long,Private
TotalValue 209	Long,Private

3.25.3.1 CurrentValue — Progress Class - Properties

The CurrentValue property is used in the Init, AddToCurrentValue 211, Calculate 211, GetCurrentValue 212, SetCurrentValue 217 and SetTotalValue 218 methods and is used to keep track of the progress. Normally it is incremented by one with each iteration of the process, but in some cases it can for example contain the byte number when processing a DOS or ASCII files using the BYTE() function.

Note that this is a **PRIVATE** property and should not be accessed outside of this class. Use GetCurrentValue 212 and SetCurrentValue 217 to retrieve and set the value respecitvely. In normal operation you never need to access this property.

It is declared as:

```
CurrentValue                Long,Private
```

3.25.3.2 DisplayControls — Progress Class - Properties

The DisplayControls property queue is used in the Init 214, AddDisplayControl 210, HideControls 214, ShowProgress 218 and the Kill 215 methods. It is used to keep track of controls that should be unhidden with the HideControls method. Note that the HideUnhide property must be set to True by the Init method for the controls to be unhidden. The queue has two properties, ITDisplayControl and UpdateOnShow.

ITDisplayControl is the FEQ value for the control.

UpdateOnShow is set to true by default by the AddDisplayControl method. It means that the control is updated (DISPLAY() is called for the control) during each call to ShowProgress 218. If the control is only to be unhidden with a call to HideControls(False) then UpdateOnShow should be set to false by setting the second parameter in AddDisplayControl to false.

It is declared as:

```
DisplayControls             &ITDisplayQueue
```

3.25.3.3 Initialized — Progress Class - Properties

This property is set to true in the Init 214 method. It is used in the ShowProgress 218, ShowUpdateProgress 219 and Update 219 methods. The purpose is to prevents problems if the Init 214

method is not called.

It is declared as:

```
Initialized                 Byte
```

3.25.3.4 HideUnhide

This property is initialized in the Init 214 method based on the value of the pHideUnhide parameter, which defaults to true. It is used in the HideControls 214 method to determine if controls should be unhidden or hidden. See Example 2 206 for an example of unhiding/hiding additional controls.

It is declared as:

```
HideUnhide                  Byte
```

3.25.3.5 PercentValue

The PercentValue property is set to 0 (zero) in the Init 214 method. It is used in the Calculate 211, GetCurrentPercent 212, SetTotalValue 218 and ShowProgress 218 methods and contains the percentage value that has been completed, ranging from 0 to 100.

Note that this is a **PRIVATE** property and should not be accessed outside of this class.

It is declared as:

```
PercentValue                Byte,Private
```

3.25.3.6 ProgressControl

The ProgressControl property is set in the Init 214 method. It is used in the GetProgressControl 213, HideControls 214 and ShowProgress 218 methods and stores the FEQ value for the progress control. It is specified in the first parameter of the Init method. Each Progress class can only control a single progress bar but you can instantiate multiple progress classes in each procedure if you need to control multiple progress bars.

Note that this is a **PRIVATE** property and should not be accessed outside of this class.

It is declared as:

```
ProgressControl             Long,Private
```

3.25.3.7 TotalValue

The TotalValue property is set in the Init 214 method. It is used in the Calculate 211, GetTotalValue 213 and SetTotalValue 218 methods and keeps the total value for the progress. Calling the SetTotalValue 218 resets all the values for a progress bar and makes it possible to reset it and re-use it without having to kill it and re-initialize it.

Note that this is a **PRIVATE** property and should not be accessed outside of this class. If you need to access it use the GetTotalValue or SetTotalValue methods.

It is declared as:

```
TotalValue                  Long,Private
```

3.25.4 Methods — Progress Class

There are currently 15 methods in the Progress Class.

```
AddDisplayControl 210       Procedure(LONG pControLToDisplay,Byte pUpdateOnShow=1)
AddToCurrentValue 211       Procedure(Long pValueToAdd),LONG,PROC ! Adds value and
returns new current value
Calculate 211               Procedure                       ! Calculates percent
GetCurrentPercent 212       Procedure(),BYTE                ! Returns 0 - 100
GetCurrentValue 212         Procedure(),LONG                ! Returns CurrentValue
GetProgressControl 213      Procedure(),LONG                ! Returns
ProgressControl
GetTotalValue 213           Procedure(),LONG                ! Returns TotalValue
HideControls 214            Procedure(BYTE pHide)           ! Hide/unhide display
controls
Init 214                    Procedure(LONG pProgressControl, Long pTotalValue,
Byte pHideUnhide=1, Byte pCanBeZeroOrOne=True)
Kill 215                    Procedure
SetCurrentValue 217         Procedure(Long pCurrentValue)
SetTotalValue 218           Procedure(LONG pTotalValue)
ShowProgress 218            Procedure
ShowUpdateProgress 219      Procedure(Long pValueToAdd)     ! Update progressbar
after adding pValueToAdd to the value.
Update 219                  Procedure ! Calls ShowUpdateProgress(1)
```

3.25.4.1 AddDisplayControl — Progress Class - Methods

Prototype: **(LONG pControLToDisplay,Byte pUpdateOnShow=1)**

pControLToDisplay FEQ of a control to display

pUpdateOnShow Indicates if the control should be updated using DISPLAY() when ShowProgress 218 is called.

The Progress Class can display multiple controls that are then updated when the progress is updated. The controls added are unhidden when the progress is initialized and hidden again if the HideUnhide 209 property is TRUE.

Example:

```
ITP  ITProgressClass
 Code
 Open(MyFile)
 Set(MYF:MyKey)
 ITP.Init(?Progress1,Records(MYF:MyKey),True) !! Initialize and tell it to handle more controls...
 ITP.AddDisplayControl(?CancelButton,False)   !! ... and add the ?CancelButton...
 ITP.HideControls(False)                      !! ... and unhide all handled controls while the progress is
```

See also:

HideUnhide 209
HideControls 214

3.25.4.2 AddToCurrentValue

Prototype: **(Long pValueToAdd),LONG,PROC**

pValueToAdd Integer value to add to the TotalValue.

Returns Returns the resulting value

This method adds the passed integer value to the TotalValue 209 property and re-calculates the progress bar percent values so when the progressbar is updated next time it will reflect the correct percentage.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:

TotalValue 209
Update 219
ShowProgress 218
[****]

3.25.4.3 Calculate

Prototype: **(none)**

This method simply calculates the PrecentValue property. You should never need to call it.

See also:

PercentValue 209
CurrentValue 208

TotalValue 209

3.25.4.4 GetCurrentPercent

Prototype: **(),BYTE**

Returns Returns the integer of the current percent of where the progress is. This number will always be in the range of 0 - 100

This method calls the Calculate 211 methods and then returns the value of the PercentValue 209 property.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End
 Message('Done with Q1, current percent is: ' & ITP.GetCurrentPercent())
 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:
PercentValue 209
Calculate 211

3.25.4.5 GetCurrentValue

Prototype: **(),LONG**

Returns Returns the integer for the current value of the progress.

This method returns the value of the TotalValue 209 property which is the destination counter to be reached at the end of the process.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
I    Long
 Code
 !! Queue is filled here.

 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 Loc:Var = ITP.GetCurrentValue()
 Display(?Loc:Var)
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End

 ITP.Kill
```

See also:
TotalValue 209
SetCurrentValue 217
GetCurrentPercent 212

3.25.4.6 GetProgressControl

Prototype: **(),LONG**

Returns Returns the FEQ of the progress bar control.

This method returns the FEQ of the progress bar control that the class is controlling.

Example:

```
ITP  ITProgressClass
 Code
 Open(MyFile)
 Set(MYF:MyKey)
 ITP.Init(?Progress1,Records(MYF:MyKey),True) !! Initialize and tell it to handle more controls...
 Message('The progress bar control FEQ is: ' & ITP.GetProgressControl())
```

See also:
Init 141
ProgressControl 209

3.25.4.7 GetTotalValue

Prototype: **(),LONG**

Returns Returns the value of the TotalValue 209 property

This method returns the value of the TotalValue 209 property.

Example:

```
ITP  ITProgressClass
 Code
 Open(MyFile)
 Set(MYF:MyKey)
 ITP.Init(?Progress1,Records(MYF:MyKey),True) !! Initialize and tell it to handle more controls...
 Message('Total value of the progress bar: ' & ITP.GetTotalValue())
 !! Message should return the same value as Records(MYF:MyKey) used to initialize the progress.
```

See also:

TotalValue 209
SetTotalValue 218
Init 214

3.25.4.8 HideControls

Prototype: **(BYTE pHide)**

pHide Indicates if the control(s) should be hidden or unhidden.

This method is used inside the class to hide and unhide the progress control and any controls that have been added to the DisplayControl 208 property queue using the AddDisplayControl 210 method. Note that the Init and Kill methods hide and unhide the controls automatically.

Example:

```
ITP  ITProgressClass
 Code
 Open(MyFile)
 Set(MYF:MyKey)
 ITP.Init(?Progress1,Records(MYF:MyKey),True) !! Initialize and tell it to handle more controls...
 ITP.AddDisplayControl(?CancelButton,False)   !! ... and add the ?CancelButton...
 ITP.HideControls(False)                      !! ... and unhide all handled controls while the progress is

 ...
 ITP.HideControls(True)                       !! ... and unhide all handled controls while the progress is
```

See also:

Init 214
Kill 215
DisplayControls 208
AddDisplayControl 210

3.25.4.9 Init

Prototype: **(LONG pProgressControl, Long pTotalValue, Byte pHideUnhide=1, Byte pCanBeZeroOrOne=True)**

pProgressControl FEQ of the progress bar control that the class updates

pTotalValue Total value to be processed

pHideUnhide Indicates if the progress bar control and any other controls in the DisplayControls 208 queue property should be unhidden when the process starts and hidden when it ends. Defaults to 1 (True)

pCanBeZeroOrOne Indicates if the pTotalValue can be 0 or 1. Defaults to True. You never need to change this setting. This was originally added because the order of the 2nd and 3rd parameters changed.

This method sets up the progress class. It sets the PROP:RangeLow and PROP:RangeHigh for the progress bar control and sets the TotalValue property. It also sets the ProgressControl 209 property and creates the DisplayControls 208 queue property. It also sets the HideControls property and finally calls the Calculate 211 method. Once the progress class is initialized you can add additional controls to be updated along with the progress bar. You call this method at the start of the progress process.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:

DisplayControls 208
ProgressControl 209
Update 219
Kill 215

3.25.4.10 Kill

Progress Class - Methods

Prototype: **(none)**

This method terminates the progress class. It calls the HideControls 214 method to hide controls that should be hidden and then disposes of the DisplayControl 208 queue property. You call this method at the end of the progress process.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
```

```
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:
Init 214
HideControls 214
DisplayControl 208

3.25.4.11 ReleaseWindow

Prototype: **(none)**

This method activates a timer based ACCEPT loop that runs for 1/100th of a second, releasing the window so it can be moved and other controls can interact. This is important to use when using the ProgressClass with tight loops that run for more than a second or two. It should called periodically during the process, depending on how long it is, perhaps every 1-5%

Example:

```
ITP  ITProgressClass
Q1   QUEUE
F1     LONG
     END
Q2   QUEUE
F2     BYTE
     END
I    LONG
P    LONG
 CODE
 !! Queues are filled here.
 ITP.Init(?Progress1,RECORDS(Q1),True)  !! Initialize with the number of records
from Q1
 ITP.AddToCurrentValue(RECORDS(Q2))     !! Add the number of records from Q2
 IF SkipQ1
   ITP.SetTotalValue(RECORDS(Q2))
 ELSE
   P = RECORDS(Q1) / 100
   LOOP I = 1 TO RECORDS(Q1)
     GET(Q1,I)
     !! Do something
     ITP.Update
```

```
    IF I % P = 0
      ITP.ReleaseWindow
    END
  END
END

LOOP I = 1 TO RECORDS(Q2)
  GET(Q2,I)
  !! Do something
  ITP.Update
END
ITP.Kill
```

See also:
Update 219

3.25.4.12 SetCurrentValue — Progress Class - Methods

Prototype: **(Long pCurrentValue)**

pCurrentValue Value to set as current value

This method sets the CurrentValue 208 property and then calls the Calculate 211 method. This method can be used to "jump" the progress bar in case it should indicate that something has been skipped. Obviously the value should be in the range of 0 to the value of the TotalValue.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 If FirstQueueShouldBeIncluded
   Loop I = 1 To Records(Q1)
     Get(Q1,I)
     !! Do something
     ITP.Update
   End
 Else
   ITP.SetCurrentValue = Records(Q1)    !! Set the current value to the end of Q1.
 End


 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:

3.25.4.13 SetTotalValue

Prototype: **(LONG pTotalValue)**

pTotalValue The new value for the TotalValue property

This method sets the TotalValue 209 property to the value passed in as pTotalValue. This can be set to adjust the total value of a process.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 If SkipQ1
   ITP.SetTotalValue(Records(Q2))
 Else
   Loop I = 1 To Records(Q1)
     Get(Q1,I)
     !! Do something
     ITP.Update
   End
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:

TotalValue 209
AddToCurrentValue 211
Init 214

3.25.4.14 ShowProgress

Prototype: **(none)**

This method does most of the work that happens when the Update 219 method is called. It recalculates the progress by calling the Calculate 211 method. It then set the PROP:Progress for the progress bar and updates any controls that need to be updated. This method is called from the ShowUpdateProgress 219 method, which in turn is called from the Update 173 method. You never need to call this method directly.

See also:
Calculate 211
Update 173
ShowUpdateProgress 219

3.25.4.15 ShowUpdateProgress

Prototype: **(Long pValueToAdd)**

pValueToAdd Indicates the value to add to the CurrentValue 208 property

This method adds the passed in value to the CurrentValue 208 property and then calls the ShowProgress 218 to update the progress bar and any related controls. Normally you do not need to call this method directly, but if you are dealing with progress bars that are not determined by a simple counter, then this is the method to call. For example if you are reading a DOS file with a 4K buffer you can use this method instead of Update and pass the size of the buffer read to it to update it correctly.

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.ShowUpdateProgress(1)
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.ShowUpdateProgress(1)
 End
 ITP.Kill
```

See also:

3.25.4.16 Update

Prototype: **(none)**

This method is one of the 3 methods that you normally need to use along with Init 214 and Kill 215. This updates the progress bar and increments the CurrentValue 208 by one (1)

Example:

```
ITP  ITProgressClass
Q1   Queue
F1     Long
     End
Q2   Queue
F2     Byte
     End
I    Long
 Code
 !! Queues are filled here.
 ITP.Init(?Progress1,Records(Q1),True)  !! Initialize with the number of records from Q1
 ITP.AddToCurrentValue(Records(Q2))     !! Add the number of records from Q2
 Loop I = 1 To Records(Q1)
   Get(Q1,I)
   !! Do something
   ITP.Update
 End

 Loop I = 1 To Records(Q2)
   Get(Q2,I)
   !! Do something
   ITP.Update
 End
 ITP.Kill
```

See also:

Init 214
Kill 215
CurrentValue 208

3.26 Record Class

3.26.1 Overview Record Class

```
ITRecordClass
CLASS(ITStringClass),TYPE,Module('ITRecordClass.clw'),Link('ITRecordClass',_ITUtilL
inkMode_),DLL(_ITUtilDllMode_)
ITRecord                    &GROUP
ITFile                      &FILE
RecSize                     Long
FieldNum                    Long
FieldNameLen                Byte
TempString                  &CString,PRIVATE
FieldFormats                &ITFieldFormats
Init                        Procedure(FILE pFile) !!, *GROUP pRecord)
AddFieldFormat              Procedure(String pFieldName, String pFormat)
GetRecordStringSize         Procedure(String pDelimeter),Long
SetFieldNameLen             Procedure(Byte pLen)
GetRecordWithData           Procedure(String pDelimeter, Byte pPad=False),String
WriteRecToFile              Procedure(String pFile, Byte pAppend)
WriteLineToFile             Procedure(String pFile, Byte pAppend, String pLine)
DisposeTempStr              Procedure
Construct                   Procedure
Destruct                    Procedure
                          End
```

3.26.2 Properties Record Class

Enter topic text here.

3.26.3 Methods Record Class

Enter topic text here.

3.27 RegistryClass

3.27.1 Overview RegistryClass

The Registry class has several very useful methods to deal with Windows Registry. Note that the some of the methods can be forces to access 32 or 64 bit keys on 64 bit systems for keys that reside in the Wow6432Node and in the 64 bit area. Essentially it allows you to access the 32 or 64bit values independently of the WoW64 subsystem and without redirection by the WoW64.

```
ITRegistryClass          CLASS(ITUtilityClass),TYPE,Module(
'ITRegistryClass.clw'),Link('ITRegistryClass.clw',_ITUtilLinkMode_),DLL(
_ITUtilDllMode_)
KeyValues                &tRegValQueue
RegistryKeys             &tRegQueue
KeyHandle                IT_HKEY
ValueStr                 &CString
ValueDW                  IT_DWORD
ValueInt64               LIKE(IT_ULARGE_INT)
ValueBuffer              &STRING !! Used for REG_BINARY, REG_EXPAND_SZ
and REG_MULTI_SZ

CloseRegistryKey         Procedure(UNSIGNED pKeyHandle)
EnumRegistrySubKeys      Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),LONG,PROC
EnumRegistryValues       Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),LONG,PROC
GetRegEx                 Procedure(LONG pRoot, STRING pKeyname, <
STRING pValuename>, <*LONG pValuetype>, BYTE  pUseBits=0),ANY
GetValueBufferSize       Procedure(UNSIGNED pKeyHandle, STRING
pValuename),LONG
GetValueType             Procedure(UNSIGNED pKeyHandle, STRING
pValuename),LONG
OpenRegistryKey          Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),UNSIGNED
PutRegEx                 Procedure(LONG pRoot, STRING pKeyname, STRING
pValuename,  STRING pValue, <LONG pValuetype>, BYTE  pUseBits=0),STRING
QueryValue               Procedure(UNSIGNED pKeyHandle, STRING
pValueName, LONG pValueType=REG_SZ)

Construct                Procedure
Destruct                 Procedure
                       END
```

3.27.2 DataTypes RegistryClass

The Registry class has two datatypes, tRegQueue and IT_ULARGE_INT.

3.27.2.1 tRegValQueue RegistryClass - DataTypes

This typed queue is used by EnumRegistryValues method that enumerates values from the specified registry key.

```
tRegValQueue            Queue,TYPE
RegistryMasterKey         LONG
RegistryKey               CSTRING(2049)
RegValueName              CSTRING(256)
RegValueDataType          LONG
RegValueString            &STRING
RegValueDW                IT_DWORD
RegValueInt64             LIKE(IT_ULARGE_INT)
                        END
```

3.27.2.2 tRegQueue — RegistryClass - DataTypes

This typed queue is used by EnumRegistrySubKeys method that enumerates sub-keys from the specified registry key.

```
tRegQueue               Queue,TYPE
RegistryMasterKey         Long
RegistryKey               CString(2049)
SubKey                    CString(256)
                        End
```

3.27.2.3 IT_ULARGE_INT — RegistryClass - DataTypes

Data structure used to handle 64bit integers in the Registry Class. It is declared as:

```
IT_ULARGE_INT           GROUP,TYPE
LowDw                     ULONG
HighDw                    ULONG
                        END
```

3.27.3 Properties — RegistryClass

The Registry Class has 7 properties:

KeyValues	&tRegValQueue
RegistryKeys	&tRegQueue
KeyHandle	IT_HKEY
ValueStr	&CString
ValueDW	IT_DWORD
ValueInt64	LIKE(IT_ULARGE_INT)
ValueBuffer	&STRING

3.27.3.1 KeyValues — RegistryClass - Properties

The RegistryValues property is used in the EnumRegistryValues and Construct methods and is used to receive registry key value information when enumerating them in the EnumRegistryValues method. This is a Queue of type tRegValQueue.

It is declared as:

```
KeyValues               &tRegValQueue
```

3.27.3.2 RegistryKeys

The RegistryKeys property is used in the EnumRegistrySubKeys 226 and Construct 230 methods and is used to receive sub-key information when enumerating them in the EnumRegistrySubKey 226 method. This is a Queue of type tRegQueue 223.

It is declared as:

```
RegistryKeys                &tRegQueue 223
```

3.27.3.3 KeyHandle

The KeyHandle property is not yet used in any methods but is reserved for future use.

It is declared as:

```
KeyHandle                   IT_HKEY
```

3.27.3.4 ValueStr

The ValueStr property is used in the GetRegEx 227, QueryValue 229 and Destruct 230 methods and is used to store values returned from QueryValue 229. It is dynamically allocated each time that QueryValue 229 is called. The memory is disposed of in the Destruct 230 method. This property is used when the value type is REG_SZ, REG_EXPAND_SZ or REG_MULTI_SZ when retrieving data from the registry.

It is declared as:

```
ValueStr                    &CString
```

3.27.3.5 ValueDW

The ValueDW property is used in the GetRegEx 227 and QueryValue 229 methods and is used to store Double Word values when the value type is REG_DWORD, REG_DWORD_LITTLE_ENDIAN or REG_DWORD_BIG_ENDIAN when retrieving data from the registry.

It is declared as:

```
ValueDW                     IT_DWORD
```

3.27.3.6 ValueInt64

The ValueInt64 property is used in the GetRegEx 227 and QueryValue 229 methods and is used to store Int64/LargeInt values when the value type is REG_QWORD when retrieving data from the registry.

It is declared as:

```
ValueInt64                  LIKE(IT_ULARGE_INT 223)
```

3.27.3.7 ValueBuffer

The ValueBuffer property is used in the GetRegEx 227 method and is used to store binary data when the value type is REG_BINARY when retrieving data from the registry.

Note: This property and the REG_BINARY is not yet implemented in the Registry Class as of April 11, 2014.

It is declared as:

```
ValueBuffer                 &String
```

3.27.4 Methods RegistryClass

The Registry Class has 11 methods:

```
CloseRegistryKey 225          Procedure(UNSIGNED pKeyHandle)
EnumRegistrySubKeys 226       Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),LONG,PROC
EnumRegistryValues 227        Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),LONG,PROC
GetRegEx 227                  Procedure(LONG pRoot, STRING pKeyname, <
STRING pValuename>, <*LONG pValuetype>, BYTE  pUseBits=0),ANY
GetValueBufferSize 228        Procedure(UNSIGNED pKeyHandle, STRING
pValuename),LONG
GetValueType 228              Procedure(UNSIGNED pKeyHandle, STRING
pValuename),LONG
OpenRegistryKey 229           Procedure(UNSIGNED pMasterKey, STRING pRegKey
, BYTE  pUseBits=0),UNSIGNED
PutRegEx 229                  Procedure(LONG pRoot, STRING pKeyname, STRING
pValuename,  STRING pValue, <LONG pValuetype>, BYTE  pUseBits=0),STRING
QueryValue 229                Procedure(UNSIGNED pKeyHandle, STRING
pValueName, LONG pValueType=REG_SZ)

Construct 230                 Procedure
Destruct 230                  Procedure
```

3.27.4.1 CloseRegistryKey

Prototype: **(UNSIGNED pKeyHandle)**

pKeyHandle Handle for the key to close

This method closes a key that has been previously opened with OpenRegistryKey 229. This method is used internally by the class.

Example:

```
ITR ITRegistryClass
K   UNSIGNED
  Code
  K = ITR.OpenRegistryKey(REG_LOCAL_MACHINE,'SOFTWARE\Icetips\Test')
  If K
```

```
    !! Valid key returned
    ITR.CloseRegistryKey(K)
  Else
    Message('Key was invalid')
  End
```

See also:
OpenRegistryKey 229

EnumRegistrySubKeys 226

3.27.4.2 EnumRegistrySubKeys

Prototype:	**(UNSIGNED pMasterKey, String pRegKey, Byte pUseBits=0),Long,PROC**
pMasterKey	The root key. Valid values are: IT_HKEY_CLASSES_ROOT, IT_HKEY_CURRENT_USER, IT_HKEY_LOCAL_MACHINE, IT_HKEY_USERS, IT_HKEY_PERFORMANCE_DATA, IT_HKEY_CURRENT_CONFIG or IT_HKEY_DYN_DATA.
pRegKey	The registry key to enumerate keys for. Example: 'SOFTWARE\SoftVelocity\Clarion8'
pUseBits	Indicates if the enumeration should be forced to use 32 or 64 bit values or default (virtualized through WoW64 subsystem) Valid values are: 0 (default), 32 (force 32bit) and 64 (force 64bit)
Returns	The method returns the number of keys enumerated, i.e. the number of records in the RegistryKey 224 property.

This method uses RegEnumKeyEx api to enumerate all subkeys from the specified key. Note that this method is not recursive, i.e. it will not enumerate sub-keys of the sub-keys. You can use the IT_HKEY_ equates or the REG_ equates from Clarion equates.clw, either will work. The IT_HKEY_ equates have the same names as the equates used by the Windows APIs except for the "IT_" prefix. This method can query both 32 and 64bit keys on 64bit operating systems. This allows you to bypass the WoW64 subsystem and query the keys directly, i.e. query 64 bit keys from a 32 bit program without the WoW64 system redirecting them to the 32 keys. This can be very helpful when accessing information stored by native 64bit programs or .NET programs that run as 64bit under 64bit operating systems.

Example:

```
ITR       ITRegistryClass
  Code
  ITR.EnumRegistrySubKeys(REG_LOCAL_MACHINE,
'SOFTWARE\Microsoft\.NETFramework'))  !! Enumerate .NET information
```

See also:
OpenRegistryKey 229

CloseRegistryKey 225

3.27.4.3 EnumRegistryValues

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.4 GetRegEx

Prototype: **(Long pRoot, String pKeyname, <String pValuename>, <*LONG pValuetype>, Byte pUseBits=0),ANY**

pRoot The root key. Valid values are: IT_HKEY_CLASSES_ROOT, IT_HKEY_CURRENT_USER, IT_HKEY_LOCAL_MACHINE, IT_HKEY_USERS, IT_HKEY_PERFORMANCE_DATA, IT_HKEY_CURRENT_CONFIG or IT_HKEY_DYN_DATA.

pKeyname The registry key to enumerate keys for. Example: 'SOFTWARE\SoftVelocity\Clarion8'

pValuename Name of the value to retrieve information from

pValuetype Optional LONG variable passed by address that will receive the datatype information.

pUseBits Indicates if the information should be fetched from the 32 or 64 bit sections of the registry or default (virtualized through WoW64 subsystem) Valid values are: 0 (default), 32 (force 32bit) and 64 (force 64bit)

Returns Returns data from the registry. This data can be a STRING, CSTRING, IT_DWORD or IT_ULARGE_INT 223

This method works for most parts exactly like the built-in GETREG function in the Clarion runtime library, with one major difference: GetRegEx can query both 32 and 64bit keys on 64bit operating systems. This allows you to bypass the WoW64 subsystem and query the keys directly, i.e. query 64 bit keys from a 32 bit program without the WoW64 system redirecting them to the 32 keys. This can be very helpful when accessing information stored by native 64bit programs or .NET programs that run as 64bit under 64bit operating systems.

Example:

```
ITR       ITRegistryClass
KeyValue  String(255)
REG       Long
  Code
```

```
  REG = REG_SZ
  KeyValue = ITR.GetRegEx(REG_LOCAL_MACHINE, 'SOFTWARE\Icetips Test',
'Test_REG_DWORD', REG, 64)
```

See also:
GetValueType 228
QueryValue 229
OpenRegistryKey 229
CloseRegistryKey 225

3.27.4.5 GetValueBufferSize

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.6 GetValueType

Prototype: **(UNSIGNED pKeyHandle, String pValuename),Long**

pKeyHandle Parameter Information
pValuename Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.7 OpenRegistryKey

Prototype: **(UNSIGNED pMasterKey, String pRegKey, Byte pUseBits=0),UNSIGNED**

pMasterKey Parameter Information
pRegKey Parameter Information
pUseBits Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.8 PutRegEx

Prototype: **(Long pRoot, String pKeyname, String pValuename, String pValue, <LONG pValuetype>, Byte pUseBits=0),STRING**

pRoot Parameter Information
pKeyname Parameter Information
pValuename Parameter Information
pValue Parameter Information
pValuetype Parameter Information
pUseBits Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.9 QueryValue

Prototype: **(UNSIGNED pKeyHandle, String pValueName, Long pValueType=REG_SZ)**

pKeyHandle Parameter Information
pValueName Parameter Information
pValueType Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.10 Construct

Prototype: **(none)**

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.27.4.11 Destruct

Prototype: **(none)**

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.28 RTF Text Class

3.28.1 Overview RTF Text Class

Enter topic text here.

3.28.2 Methods RTF Text Class

Enter topic text here.

3.28.3 Properties RTF Text Class

Enter topic text here.

3.29 Select List Class

3.29.1 Overview Select List Class

This class handles selections in a listbox.

```
ITSelectListClass
CLASS(ITUtilityClass),TYPE,Module('ITSelectListClass.clw'),Link('ITSelectListClass'
,_ITUtilLinkMode_),DLL(_ITUtilDllMode_)

ListFEQ                   Long
CurrentChoice             Long
HighPtr                   Long
LowPtr                    Long
LastChoice                Long
MarkField                 ANY
MarkQueue                 &QUEUE
Init                      Procedure(Long pListFeq, *? pMarkField, QUEUE pMarkQueue)
Kill                      Procedure
TakeNewSelection          Procedure(),BYTE
                        END
```

3.29.2 Properties Select List Class

Enter topic text here.

3.29.3 Methods Select List Class

Enter topic text here.

3.30 SetupBuilder Class

3.30.1 Overview SetupBuilder Class

The SetupBuilder Class (ITSetupBuilderClass) contains various methods to make working with SetupBuilder installs easier. Several methods are not really SetupBuilder specific and could be used with other installation software as well. This particularly applies to methods that copy files from a "global" location to a "local" location. A Global location is a path that is accecssible by all users under all operating systems. A Local location is a path that is accessible only by the current user.

The SetupBuilder class has two main type of methods. The first one serves in your application to copy files from a globally accessible location where SetupBuilder installs the files, to a locally accessible location where your application uses them. This is particularly useful under **Windows Vista** where the installer can only install files into folders that are open to all users. That location may not be appropriate, or even visible, for the users of your application, i.e. if there is user specific data. These methods copy the data from the global folder to a local folder and update a registry key so you can check if the data has been copied.

The second type are methods that you can use to call SetupBuilder to compile your projects, update compiler variables and determine destination folder for the compiled install and build report. Please note that you must have SetupBuilder version 6.6 build 2000 or later for this to work properly. Previous versions will be able to compile the project, but can not change the output folder or the name of the compiled install executable.

Check out the short tutorial 254 at the end of this chapter.

Please check the **TestSetupBuilderClass** procedure in the **UtilDemo.app** that is located in your "Clarion\3rdParty\Examples\ITUtilities" folder. It allows you to pick a source and destination CSIDLs for the copying part of the class, as well as pick a project and destination for compiling your project.

```
ITSetupBuilderClass
CLASS(ITVersionClass),TYPE,Module('ITSetupBuilderClass.clw'),|

Link('ITSetupBuilderClass',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
CompilerVariables 236          &SBCompileVars
DestinationFolder 236          CString(2049)
FilesCopied 237                Byte
GlobalCSIDL 237                Long
LocalCSIDL 237                 Long
PathString 238                 CString(1025)  ! Path for CSIDL_COMMON, CSIDL_LOCAL,
HKLM and HKCU
SBBuildNumber 238              Long
SBCommandLine 239              &CString
SBErrorLogFile 239             CString(2049)  ! Error log file if an error occurs
                                              ! during compile. If no error occurs,
                                              ! this file does not exist.
SBExecutable 239               CString(2049)  ! Setupbuilder.exe path and name for
                                              ! calling compiler
SBGlobalInstallPath 240        CString(2049)  ! CSIDL_COMMON
SBGlobalRegistryKey 240        CString(2049)  ! HKLM
SBHtmlLogFile 240              CString(2049)  ! The HTML file created after a
                                              ! compile
SBLocalInstallPath 240         CString(2049)  ! CSIDL_LOCAL
SBLocalRegistryKey 241         CString(2049)  ! HKCU
SBMajorVersion 241             Byte
SBMinorVersion 241             Byte
```

```
SBProjectToCompile 241          CString(2049)  ! Project name and path.

AddCompilerVariable 242         Procedure(String pVariable, |
                                        String pValue, |
                                        Byte pQuoteValue=0)
BuildCommandLine 243            Procedure(String pBaseCommandLine),String
CompileSBProject 243            Procedure(String pProjectToCompile),Long,PROC
CopyTheFiles 245                Procedure(),Long,PROC   ! Returns number of files
copied
CreateDestinationFolder 245     Procedure(),Long,PROC   ! Creates the Destination
Folder tree
FinishInstall 246               Procedure(String pPath, Long pLocal, Long
pGlobal),Long,PROC
GetDestinationFolder 246        Procedure(),String,PROC ! Fills DestinationFolder and
returns it.
GetGlobalKey 247                Procedure(),String
GetGlobalPath 247               Procedure(),String
GetLocalKey 247                 Procedure(),String
GetLocalPath 248                Procedure(),String
GetSBExecutable 248             Procedure(),BYTE        ! Returns true if executable
                                                      ! is found
GetSBVersionInformation 248     Procedurex
SetDestinationFolder 249        Procedure(String pDestination)
SetGlobalCSIDL 249              Procedure(Long pCSIDL)
SetLocalCSIDL 250               Procedure(Long pCSIDL)
SetPathString 250               Procedure(String pPathString)
ShowHTMLLogFile 251             Procedure
ShowLogFile 252                 Procedure(Byte pOpenInWindow=True)
ShowLogFile 252                 Procedure(INIClass pINIMgr)
Construct 253                   Procedure
Destruct 254                    Procedure
                        END
```

3.30.2 Data Types — SetupBuilder Class

The SetupBuilder class uses one special data type for compiler variables.

SBCompileVars 235

See also:
CompilerVariables 236
AddCompilerVariable 242

3.30.2.1 SBCompileVars — SetupBuilder Class - Data Types

The **SBCompileVars** is a queue type that is used in the SetupBuilder class to store compiler variables and new values to update those variables with during the compiling process.

```
SBCompileVars           QUEUE,TYPE
VariableName              CString(101)
VariableValue             CString(2049)
QuoteValue                Byte
                        END
```

See also:

CompilerVariables 236

3.30.3 Properties SetupBuilder Class

The SetupBuilder class currently has 18 properties that store various information during the compile process. Also when the class is used to finish an install and copy files.

```
CompilerVariables 236          &SBCompileVars 235
DestinationFolder 236          CString(2049)
FilesCopied 237                Byte
GlobalCSIDL 237                Long
LocalCSIDL 237                 Long
PathString 238                 CString(1025)  ! Path for CSIDL_COMMON, CSIDL_LOCAL,
HKLM and HKCU
SBBuildNumber 238              Long
SBCommandLine 239              &CString
SBErrorLogFile 239             CString(2049)  ! Error log file if an error occurs
during compile
SBExecutable 239               CString(2049)  ! Setupbuilder.exe path and name for
calling compiler
SBGlobalInstallPath 240        CString(2049)  ! CSIDL_COMMON
SBGlobalRegistryKey 240        CString(2049)  ! HKLM
SBHtmlLogFile 240              CString(2049)  ! The HTML file created after a compile
SBLocalInstallPath 240         CString(2049)  ! CSIDL_LOCAL
SBLocalRegistryKey 241         CString(2049)  ! HKCU
SBMajorVersion 241             Byte
SBMinorVersion 241             Byte
SBProjectToCompile 241         CString(2049)  ! Project name and path.
```

3.30.3.1 CompilerVariables SetupBuilder Class - Properties

Queue used to store Compiler variables and new values to update the SetupBuilder project script with. It is declared as:

```
CompilerVariables          &SBCompileVars 235
```

See also:
SBCompileVars 235
AddCompilerVariable 242

3.30.3.2 DestinationFolder SetupBuilder Class - Properties

The DestinationFolder property is a Cstring that contains the destination folder for the compiled install and Build Report when the SetupBuilder compiler is called. By default SetupBuilder uses the path where the project file (*.sb6) is and creates a subfolder with the same name as the project file. For example, if the project file is located in "C:\Products\MyProduct\MyProduct.sb6" the SetupBuilder will compile the install and build report into "C:\Products\MyProduct\MyProduct" which is the default destination folder. The DestinationFolder can be changed to any accessible folder location. If the DestinationFolder does not exist, the class will create it inside the CompileSBProject 243 method before the compiler is called to create the install.

DestinationFolder CString(2049)

See also:
CompileSBProject 243
GetDestinationFolder 246
SetDestinationFolder 249

3.30.3.3 FilesCopied

This property is a Byte variable that indicates if the copy process from the Global folder to the Local folder has been done. This property is set in SetPathString 250 and FinishInstall 246 and checked and updated in the CopyTheFiles 245 method.

FilesCopied Byte

3.30.3.4 GlobalCSIDL

This Long property contains the CSIDL value for the Global folder that files should be copied FROM. This could for example be IT_CSIDL_COMMON_APPDATA.

GlobalCSIDL Long

See also:
LocalCSIDL 237
SetGlobalCSIDL 249
FinishInstall 246

3.30.3.5 LocalCSIDL

This Long property contains the CSIDL value for the Local folder that files should be copied TO. This could for example be IT_CSIDL_PERSONAL or IT_CSIDL_LOCAL_APPDATA depending on the nature of the files.

If this is application data, that the user should not mess with, IT_CSIDL_LOCAL_APPDATA might be more appropriate. Note that on Windows Vista, IT_CSIDL_LOCAL_APPDATA is hidden so it does not show up in the Explorer window.

If this is data that the user may need to copy, backup, etc. then IT_CSIDL_PERSONAL would be more appropriate. IT_CSIDL_PERSONAL generally refers to the "My Documents" folder. Please check out our FREE SpecialFolder *(http://www.icetips.com/downloadfile.php?FileID=71)* program that is invaluable in determining the actual locations on the various operating systems.

LocalCSIDL Long

See also:
SetLocalCSIDL 250

FinishInstall 246

3.30.3.6 PathString

This Cstring property is used to determine the Global and Local locations of the files for copying as well as updating to the registry keys. This property contains a partial path that is appended to the CSIDL paths pointed to by GlobalCSIDL 237 and LocalCSIDL 237. It is also used as key for the HKLM and HKCU registry keys to store the value of the FilesCopied property.

For example if the GlobalCSIDL is IT_CSIDL_COMMON_APPDATA resulting in "C:\Documents and Settings\All Users\Application Data" and the LocalCSIDL is IT_CSIDL_PERSONAL resulting in "C:\Documents and Settings\Arnor\My Documents" and the PathString is "Icetips Creative\Utilities", then this would translate to:

Folders:
SBGlobalInstallPath = "C:\Documents and Settings\All Users\Application Data\Icetips Creative\Utilities"
SBLocalInstallPath = "C:\Documents and Settings\Arnor\My Documents\Icetips Creative\Utilities"

Registry keys:
HKLM\SOFTWARE\Icetips Creative\Utilities
HKCU\SOFTWARE\Icetips Creative\Utilities

The HKCU (Current User) registry keys would contain a single value, FilesCopied, which would be either true or false depending on if the files have been copied or not. The HKLM (Local Machine) registry key is not used at this point.

```
PathString              CString(1025)
```

See also:
SetPathString 250
SBGlobalInstallPath 240
SBGlobalRegistryKey 240
SBLocalInstallPath 240
SBLocalRegistryKey 241
GetGlobalPath 247
GetGlobalKey 247
GetLocalPath 248
GetLocalKey 247
SetGlobalCSIDL 249
SetLocalCSIDL 250

3.30.3.7 SBBuildNumber

This Long property contains the build number of the SetupBuilder executable. For example in version 6.6.2000, the SBBuildNumber would be 2000. This is retrieved from the version information in the SetupBuilder executable, SBExecutable 239.

```
SBBuildNumber           Long
```

See also:
GetSBExecutable 248

GetSBVersionInformation 248
SBExecutable 239

3.30.3.8 SBCommandLine

SetupBuilder Class - Properties

This dynamic CString contains the commandline used to run the SetupBuilder compiler. It is constructed at runtime based on the executable found for the project file, compiler variables 236 added to the project and if destination is specified or not. As different versions of Windows have different maximum sizes for the command line, the class checks to see if the commandline being constructed is larger than the maximum allowed by the operating system. If it is too big, the CompileSBProject 243 method will return -2 and the SBCommandLine 239 will be empty. The SBCommandLine 239 is disposed in the Destruct 254 method.

SBCommandLine &CString

See also:
AddCompilerVariable 242
CompileSBProject 243
CompilerVariables 236
Destruct 254

3.30.3.9 SBErrorLogFile

SetupBuilder Class - Properties

If an error occurs during the SetupBuilder compilation, an error log file is created and it's location is placed in this CString property. The location depends on what version of SetupBuilder is being used. If no error occurs during compiling, this file does not exist. I.e. it only exists after a failed compile. There are two methods that can display the error log file. The demo program shows how to determine if an error has occurred and open the log file in a window.

SBErrorLogFile CString(2049)

See also:
ShowLogFile - Window 252
ShowLogFile - ShellExecute 253

3.30.3.10 SBExecutable

SetupBuilder Class - Properties

This CString property contains the full path name to the SetupBuilder executable file. The file location is found by using file association or registry settings.

SBExecutable CString(2049)

See also:
GetSBExecutable 248

3.30.3.11 SBGlobalInstallPath

This CString property contains the Global path constructed in the SetGlobalCSIDL 250 and is used as the SOURCE folder for the CopyTheFiles 245 method.

SBGlobalInstallPath CString(2049)

See also:
CopyTheFiles 245
GetGlobalPath 247
SBLocalInstallPath 240
SetGlobalCSIDL 249

3.30.3.12 SBGlobalRegistryKey

This CString property contains the Global registry key. This key is always under the HKLM\SOFTWARE node in the registry. In the current release (Beta 3) this key is not used by the class, except it is constructed properly in the SetPathString 250 method and returned by the GetGlobalKey 247 method.

SBGlobalRegistryKey CString(2049)

See also:
GetGlobalKey 247
SetPathString 250

3.30.3.13 SBHtmlLogFile

This CString property contains the name of the HTML Build Report that SetupBuilder creates when the compiling is done. It is always stored with the compiled install executable with the same name, but a ".htm" extension.

SBHtmlLogFile CString(2049)

See also:
ShowHTMLLogFile 251

3.30.3.14 SBLocalInstallPath

This CString property contains the Local path constructed in the SetLocalCSIDL 250 and is used as the DESTINATION folder for the CopyTheFiles 245 method.

SBLocalInstallPath CString(2049)

See also:
CopyTheFiles 245
GetLocalPath 248
SBGlobalInstallPath 240

SetLocalCSIDL 250

3.30.3.15 SBLocalRegistryKey

This CString property contains the Local registry key that will be updated with the FilesCopied value. This key is always under the HKCU\SOFTWARE node in the registry. The value of this property is constructed in the SetPathString 250.

```
SBLocalRegistryKey        CString(2049)
```

See also:
GetLocalKey 247
SetPathString 250

3.30.3.16 SBMajorVersion

This Byte property contains the Major version. For example in version 6.6.2000, the SBMajorVersion would be 6. In 6.5.1711 it would also be 6.

```
SBMajorVersion            Byte
```

See also:
GetSBVersionInformation 248
SBBuildNumber 238
SBMinorVersion 241

3.30.3.17 SBMinorVersion

This Byte property contains the Minor version. For example in version 6.6.2000, the SBMinorVersion would be 6. In 6.5.1711 it would be 5.

```
SBMinorVersion            Byte
```

See also:
GetSBVersionInformation 248
SBBuildNumber 238
SBMajorVersion 241

3.30.3.18 SBProjectToCompile

This CString property contains the name of the project to compile. This must be fully qualified path and filename and must have a .SB5 or .SB6 extension.

```
SBProjectToCompile        CString(2049)
```

See also:

CompileSBProject 243

3.30.4 Methods SetupBuilder Class

The SetupBuilder class has currently 24 methods including the constructor and destructor.

```
AddCompilerVariable 242      Procedure(String pVariable, |
                                     String pValue, |
                                     Byte pQuoteValue=0)
BuildCommandLine 243         Procedure(String pBaseCommandLine),String
CompileSBProject 243         Procedure(String pProjectToCompile),Long,PROC
CopyTheFiles 245             Procedure(),Long,PROC
CreateDestinationFolder 245  Procedure(),Long,PROC
FinishInstall 246            Procedure(String pPath,
                                     Long pLocal,
                                     Long pGlobal),Long,PROC
GetDestinationFolder 246     Procedure(),String,PROC
GetGlobalKey 247             Procedure(),String
GetGlobalPath 247            Procedure(),String
GetLocalKey 247              Procedure(),String
GetLocalPath 248             Procedure(),String
GetSBExecutable 248          Procedure(),BYTE
GetSBVersionInformation 248  Procedure
SetDestinationFolder 249     Procedure(String pDestination)
SetGlobalCSIDL 249           Procedure(Long pCSIDL)
SetLocalCSIDL 250            Procedure(Long pCSIDL)
SetPathString 250            Procedure(String pPathString)
ShowHTMLLogFile 251          Procedure
ShowLogFile 252              Procedure(Byte pOpenInWindow=True)
ShowLogFile 252              Procedure(INIClass pINIMgr)

Construct 253                Procedure
Destruct 254                 Procedure
```

3.30.4.1 AddCompilerVariable SetupBuilder Class - Methods

Prototype: **(String pVariable, String pValue, Byte pQuoteValue=0)**

pVariable The name of the Compiler variable to update during compile.

pValue The value to pass to SetupBuilder for this variable.

pQuoteValue Determines if the pValue should be quoted when added to the command line.

Returns The method does not return a value

This method is used before compiling a SetupBuilder project. It allows you to pass data to the SetupBuilder compiler that will update the compiler variables in the project with the data in the pValue. This is placed into the CompilerVariables 236 queue and then used in the BuildCommandLine 243 to construct the correct command line to pass to the SetupBuilder compiler.

Please note that the command line length is limited and varies depending on what operating system is in use. There is a basic length checking mechanism built into the BuildCommandLine 243 method and if the command line exceeds the maximum allowed by the operating system, the CompileSBProject 243

terminates and returns -2 to the calling code.

Example:

```
ITS.AddCompilerVariable('PRODUCTVER','0.95.000',True)
ITS.AddCompilerVariable('EXENAME','TestBuild_0.95.000.EXE',True)
```

See also:
BuildCommandLine 243
CompileSBProject 243
CompilerVariables 236

3.30.4.2 BuildCommandLine

SetupBuilder Class - Methods

Prototype: **(String pBaseCommandLine),String**

pBaseCommandLine Contains the path and name of the SetupBuilder executable and the path and the name of the SetupBuilder project to compile.

Returns Returns the full command line to execute

This method builds up a command line to compile the project. The basic command line that is passed to it contains the SetupBuilder executable and the project to compile, like:

ShortPath(SELF.SBExecutable) & ' /C "' & ShortPath(SELF.SBProjectToCompile) & ""

Optionally compiler variables can be added to the commandline by using the AddCompilerVariable 242 method. A destination folder can also be specified, but it is optional. If no destination folder is specified, it is constructed by using the project file path and name in the same manner as SetupBuilder does it. This method is called automatically by the CompileSBProject 243.

Please note that the command line length is limited and varies depending on what operating system is in use. There is a basic length checking mechanism built into this method and if the command line exceeds the maximum allowed by the operating system, the CompileSBProject 243 terminates and returns -2 to the calling code.

See also:
AddCompilerVariable 242
CompileSBProject 243

3.30.4.3 CompileSBProject

SetupBuilder Class - Methods

Prototype: **(String pProjectToCompile),Long,PROC**

pProjectToCompile Path and name of the SetupBuilder project (*.sb5 or *.sb6 files) to compile.

Returns True if the project compiled.
False if the project failed.
-1 if the SetupBuilder executable could not be found.
-2 if the command line is too big.

This method takes the project file passed to it and executes the SetupBuilder compiler to compile it. The example below shows the code that is in the TestSetupBuilderClass proceudure in the UtilDemo.app example application in your "Clarion\3rdParty\Examples\ITUtilities" folder.

Example:

```
CompileTheProject           ROUTINE
 Data
R  Long
 Code
 If Loc:SBProjectDestFolder
   ! Set the destination folder if it is specified.  Comment this out if you
   ! want to use the SetupBuilder default destination.
   ITS.SetDestinationFolder(Loc:SBProjectDestFolder)
 End

 ! First parameter is the SB Compiler Variable.
 ! Second parameter is the value to place into the compiler variable
 ! Third parameter determines if the VALUE is enclosed in double quotes or not.
 ITS.AddCompilerVariable('PRODUCTVER','0.95.000',True)
 ITS.AddCompilerVariable('EXENAME','TestBuild_0.95.000.EXE',True)

 ! NOTE:  This information is sent via the command line to SetupBuilder.
 !        The length of the command line is limited, depending on what
 !        Operating System you are using.  The class will not run the
 !        compiler if the command line exeeds the OS limit, avoiding
 !        making a mess of things.  In order to keep the command line
 !        as short as possible, make sure that you use ShortPath()
 !        on any paths that you need to pass to the SB compiler.

 ! Compile the project
 R =  ITS.CompileSBProject(Loc:SBProjectToCompile)
 ITS.ODS('Return value: ' & R)
 Case R
 Of 1
   If Message('The project was compiled successfully.  ' &|
              'Do you want to view the Build HTML file?',|
              'Project compiled successfully',ICON:Exclamation,|
               BUTTON:No+BUTTON:Yes,BUTTON:Yes) = BUTTON:Yes
     ITS.ShowHTMLLogFile
   End
 Of 0
   If Message('Error occured while compiling the project.  ' &|
              'Do you want to view the compile log?',|
              'Errors occurred',ICON:Hand,|
              BUTTON:Yes+BUTTON:No,BUTTON:Yes) = BUTTON:Yes
     ITS.ShowLogFile(IniMGR)
   End
 Of -1
   Message('Both SetupBuilder 5 and SetupBuilder 6 are installed ' &|
           'on this machine.  In that case only SetupBuilder 6.x is ' &|
           'supported and a SetupBuilder 5.x project can not be compiled.',|
           'SetupBuilder 5 and 6 detected',ICON:Hand)
 Of -2
   Message('The Command line was too long.',|
           'Command Line is too long',ICON:Hand)

 End
```

See also:

AddCompilerVariable 242
BuildCommandLine 243

3.30.4.4 CopyTheFiles

Prototype: **(),Long,PROC**

Returns Number of files copied

This method creates a directory structure specified in the PathString property in the folder specified by the LocalCSIDL property. It uses the CreateDirectories method from the UtilityClass and the CopyFiles method from the ShellClass to copy the folder structure from the Global folder in the SBGlobalInstallPath property to the Local folder in the SBLocalInstallPath property. This method is called by the FinishInstall method. When the method is finished copying, it updates the FilesCopied value of the HKCU\SOFTWARE registry key along with the partial path that is specfied in the PathString 238 property.

Example:

```
CopyTheFiles            ROUTINE

 If Loc:LocalCSIDLNumber And Loc:GlobalCSIDLNumber
   ITS.FinishInstall(Loc:PartialPath, Loc:LocalCSIDLNumber, Loc:GlobalCSIDLNumber)
 End
```

See also:
GlobalCSIDL 237
LocalCSIDL 237
PathString 238
SBGlobalInstallPath 240
SBLocalInstallPath 240

CreateDirectories 342
UtilityClass 335
ShellClass 257

3.30.4.5 CreateDestinationFolder

Prototype: **(),Long,PROC**

Returns The number of directories created

This method uses the CreateDirectories 342 method from the Utilities class 335 to create the destination folder tree specified in the DestinationFolder 236 property. This method is used by the CompileSBProject 243 method.

Example:

```
 ITS.CreateDestinationFolder
```

See also:
CompileSBProject 243
CreateDirectories 342
DestinationFolder 236
Utilities class 335

3.30.4.6 FinishInstall

Prototype: **(String pPath, Long pLocal, Long pGlobal),Long,PROC**

pPath Partial path

pLocal Local CSIDL value, such as IT_CSIDL_PERSONAL

pGlobal Global CSIDL value, such as IT_CSIDL_COMMON_APPDATA

Returns Returns number of files copied

This mehod is really the only method that you need to use to set up copying files from one place to the other. See the CopyTheFiles routine in the TestSetupBuilderClass procedure in the UtilDemo.app in your "Clarion\3rdParty\Examples\ITUtilities" folder.

Example:

```
CopyTheFiles            ROUTINE

 If Loc:LocalCSIDLNumber And Loc:GlobalCSIDLNumber
   ITS.FinishInstall(Loc:PartialPath,Loc:LocalCSIDLNumber,Loc:GlobalCSIDLNumber)
 End
```

See also:
CopyTheFiles 245
SetGlobalCSIDL 249
SetLocalCSIDL 250
SetPathString 250

3.30.4.7 GetDestinationFolder

Prototype: **(),String,PROC**

Returns Returns the contents of the DestinationFolder property

If the DestinationFolder 236 property is not set, this method will construct it based on the path and name of the Project file that is being compiled, stored in the SBProjectToCompile 241 property

Example:

```
S = ITS.GetDestinationFolder()
```

See also:
DestinationFolder 236
SBProjectToCompile 241

3.30.4.8 GetGlobalKey

Prototype: **(),String**

Returns Returns the contents of the SBGlobalRegistryKey property.

This method returns the contents of the SBGlobalRegistryKey 240 property.

Example:

```
K = ITS.GetGlobalKey()
```

See also:
SBGlobalRegistryKey 240

3.30.4.9 GetGlobalPath

Prototype: **(),String**

Returns Returns the contents of the SBGlobalInstallPath property.

This method returns the contents of the SBGlobalInstallPath 240 property.

Example:

```
P = ITS.GetGlobalPath()
```

See also:
SBGlobalInstallPath 240

3.30.4.10 GetLocalKey

Prototype: **(),String**

Returns Returns the contents of the SBLocalRegistryKey property

This method returns the contents of the SBLocalRegistryKey 241 property

Example:

```
K = ITS.GetLocalKey()
```

See also:
SBLocalRegistryKey 241

3.30.4.11 GetLocalPath

Prototype: **(),String**

Returns Returns the contents of the SBLocalInstallPath property.

This method returns the contents of the SBLocalInstallPath 240 property.

Example:

```
P = ITS.GetLocalPath()
```

See also:
SBLocalInstallPath 240

3.30.4.12 GetSBExecutable

Prototype: **(),BYTE**

Returns Returns true if the SetupBuilder executable was found

This method finds the executable for the appropriate version of SetupBuilder. This is determined by the extension of the Project to compile stored in the SBProjectToCompile 241 property. This method also constructs the path and filename of the error log, stored in the SBErrorLogFile 239 property. This method is used by the CompileSBProject 243 method. The path and name of the SetupBuilder executable is stored in the SBExecutable 239 property.

Example:

```
If ITS.GetSBExecutable()
  Run(ITS.SBExecutable)
End
```

See also:
CompileSBProject 243
SBExecutable 239
SBProjectToCompile 241

3.30.4.13 GetSBVersionInformation

Prototype: **None**

Returns The method does not return a value

This method retrieves the version information from the SetupBuiler executable stored in the SBExecutable 239 property. It then splits up the first 3 components into SBMajorVersion 241, SBMinorVersion 241 and SBBuildNumber 238 properties. The method uses the GetVersionInfo method of the Version Class 355 to retrieve the version information.

Example:

```
 ITS.GetSBVersionInformation
 Message('SetupBuilder version: ' & ITS.SBMajorVersion & '.' & ITS.SBMinorVersion &
'.' & ITS.SBBuildNumber)
```

See also:
SBBuildNumber 238
SBExecutable 239
SBMajorVersion 241
SBMinorVersion 241
Version Class 355

3.30.4.14 SetDestinationFolder

Prototype: **(String pDestination)**

pDestination The Destination folder

Returns The method does not return a value

The method assigs the pDestination parameter to the DestinationFolder 236 property. This property is used to determine the folder where the compiled installation executable and build report html files are put after the project is compiled with the CompileSBProject 243 method. To specify a non-default destination, call this method before calling the CompileSBProject 243 method.

Example:

```
If Loc:SBProjectDestFolder
  ! Set the destination folder if it is specified.
  ITS.SetDestinationFolder(Loc:SBProjectDestFolder)
End
! Compile the project
R =  ITS.CompileSBProject(Loc:SBProjectToCompile)
```

See also:
DestinationFolder 236
CompileSBProject 243

3.30.4.15 SetGlobalCSIDL

Prototype: **(Long pCSIDL)**

pCSIDL The CSIDL value to use

Returns The method does not return a value

This method constructs a path name based on the CSIDL passed to it and the PathString 238 property using the GetSpecialFolder method of the Shell Class 257. The CSIDL value is stored in the GlobalCSIDL 237 property.

Example:

```
GetGlobalPath           ROUTINE
 Get(SFQ,Choice(?Loc:GlobalCSIDL))
```

```
ITS.SetPathString(Loc:PartialPath)
ITS.SetGlobalCSIDL(SFQ.FolderIDValue)

Loc:GlobalCSIDLNumber = SFQ.FolderIDValue
Loc:GlobalPath        = ITS.GetGlobalPath()
Loc:GlobalKey         = 'HKLM:\' & ITS.GetGlobalKey()
Display()
```

See also:
GlobalCSIDL 237
PathString 238
Shell Class 257

3.30.4.16 SetLocalCSIDL

SetupBuilder Class - Methods

Prototype: **(Long pCSIDL)**

pCSIDL The CSIDL value to use

Returns The method does not return a value

This method constructs a path name based on the CSIDL passed to it and the PathString 238 property using the GetSpecialFolder method of the Shell Class 257. The CSIDL value is stored in the LocalCSIDL 237 property.

Example:

```
GetLocalPath            ROUTINE
 Get(SFQ,Choice(?Loc:LocalCSIDL))
 ITS.SetPathString(Loc:PartialPath)
 ITS.SetLocalCSIDL(SFQ.FolderIDValue)

 Loc:LocalPath        = ITS.GetLocalPath()
 Loc:LocalKey         = 'HKCU:\' & ITS.GetLocalKey()
 Loc:LocalCSIDLNumber = SFQ.FolderIDValue
 Display()
```

See also:
LocalCSIDL 237
PathString 238
Shell Class 257

3.30.4.17 SetPathString

SetupBuilder Class - Methods

Prototype: **(String pPathString)**

pPathString Partial path to add to Global and Local folders as well as registry key

Returns The method does not return a value.

This method assigns the pPathString parameter value to the PathString 238 property. It also constructs the SBGlobalRegistryKey 240 and SBLocalRegistryKey 241 properties from the PathString. Finally it

assigns the appropriate HKCU registry key to the FilesCopied 237 property indicating if the files have been copied to the local folder for the current user or not.

Example:

```
GetLocalPath            ROUTINE
 Get(SFQ,Choice(?Loc:LocalCSIDL))
 ITS.SetPathString(Loc:PartialPath)
 ITS.SetLocalCSIDL(SFQ.FolderIDValue)

 Loc:LocalPath        = ITS.GetLocalPath()
 Loc:LocalKey         = 'HKCU:\' & ITS.GetLocalKey()
 Loc:LocalCSIDLNumber = SFQ.FolderIDValue
 Display()
```

See also:
FilesCopied 237
PathString 238
SBGlobalRegistryKey 240
SBLocalRegistryKey 241

3.30.4.18 ShowHTMLLogFile

SetupBuilder Class - Methods

Prototype: **None**

Returns The method does not return a value.

This method uses the OpenURL method of the Shell Class 257 to open the Build Report html file. The location and name of the Build Report is stored in the SBHtmlLogFile 240 property. The location is retrieved in the CompileSBProject 243 method and also in the BuildCommandLine 243 method.

Example:

```
R =  ITS.CompileSBProject(Loc:SBProjectToCompile)
ITS.ODS('Return value: ' & R)
Case R
Of 1
  If Message('The project was compiled successfully.  ' &|
             'Do you want to view the Build HTML file?',|
             'Project compiled successfully',ICON:Exclamation,|
              BUTTON:No+BUTTON:Yes,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowHTMLLogFile
  End
Of 0
  If Message('Error occured while compiling the project.  ' &|
             'Do you want to view the compile log?',|
             'Errors occurred',ICON:Hand,|
             BUTTON:Yes+BUTTON:No,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowLogFile(IniMGR)
  End
Of -1
  Message('Both SetupBuilder 5 and SetupBuilder 6 are installed ' &|
          'on this machine.  In that case only SetupBuilder 6.x is ' &|
          'supported and a SetupBuilder 5.x project can not be compiled.',|
          'SetupBuilder 5 and 6 detected',ICON:Hand)
Of -2
  Message('The Command line was too long.',|
          'Command Line is too long',ICON:Hand)
```

```
End
```

See also:
BuildCommandLine 243
CompileSBProject 243
SBHtmlLogFile 240
Shell Class 257

3.30.4.19 ShowLogFile - Window

Prototype: **(INIClass pINIMgr)**

pINIMgr Global INI Manager class that is used to store window size and position.

Returns The method does not return a value

This method can be called to open and view the error log file generated if the compile process failed. That happens only if the CompileSBProject 243 method returns false. This method shows the file in a Clarion window that will save it's size and position using the standard INI Manager class being passed in the pINIMgr parameter. There is another ShowLogFile 252 method that can show the error log in either a window (which does not store it's size and position) or using the associated program by using ShellExecute. The filename for the error log file is stored in the SBErrorLogFile 239 property.

Example:

```
R =  ITS.CompileSBProject(Loc:SBProjectToCompile)
Case R
Of 1
  If Message('The project was compiled successfully.  ' &|
             'Do you want to view the Build HTML file?',|
             'Project compiled successfully',ICON:Exclamation,|
              BUTTON:No+BUTTON:Yes,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowHTMLLogFile
  End
Of 0
  If Message('Error occured while compiling the project.  ' &|
             'Do you want to view the compile log?',|
             'Errors occurred',ICON:Hand,|
             BUTTON:Yes+BUTTON:No,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowLogFile(IniMGR)
  End
Of -1
  Message('Both SetupBuilder 5 and SetupBuilder 6 are installed ' &|
          'on this machine.  In that case only SetupBuilder 6.x is ' &|
          'supported and a SetupBuilder 5.x project can not be compiled.',|
          'SetupBuilder 5 and 6 detected',ICON:Hand)
Of -2
  Message('The Command line was too long.',|
          'Command Line is too long',ICON:Hand)

End
```

See also:
CompileSBProject 243
SBErrorLogFile 239
ShowLogFile - ShellExecute 252

3.30.4.20 ShowLogFile - ShellExecute

SetupBuilder Class - Methods

Prototype: **(Byte pOpenInWindow=True)**

pOpenInWindow Indicates if the logfile should be opened in a Clarion window or if it should be opened with associated program using ShellExecute.

Returns The method does not return a value

This method can be called to open and view the error log file generated if the compile process failed. That happens only if the CompileSBProject 243 method returns false. There is another ShowLogFile 252 method that will always show the error log in a Clarion window which is resizable and stores the window size and location in the standard INIManager class, which is passed to it. The filename for the error log file is stored in the SBErrorLogFile 239 property.

Example:

```
R =  ITS.CompileSBProject(Loc:SBProjectToCompile)
Case R
Of 1
  If Message('The project was compiled successfully.  ' &|
             'Do you want to view the Build HTML file?',|
             'Project compiled successfully',ICON:Exclamation,|
              BUTTON:No+BUTTON:Yes,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowHTMLLogFile
  End
Of 0
  If Message('Error occured while compiling the project.  ' &|
             'Do you want to view the compile log?',|
             'Errors occurred',ICON:Hand,|
             BUTTON:Yes+BUTTON:No,BUTTON:Yes) = BUTTON:Yes
    ITS.ShowLogFile(True)    ! Shows in Clarion window
    !ITS.ShowLogFile(False)  ! Would use associated program
  End
Of -1
  Message('Both SetupBuilder 5 and SetupBuilder 6 are installed ' &|
          'on this machine.  In that case only SetupBuilder 6.x is ' &|
          'supported and a SetupBuilder 5.x project can not be compiled.',|
          'SetupBuilder 5 and 6 detected',ICON:Hand)
Of -2
  Message('The Command line was too long.',|
          'Command Line is too long',ICON:Hand)

End
```

See also:
CompileSBProject 243
SBErrorLogFile 239
ShowLogFile - Window 252

3.30.4.21 Construct

SetupBuilder Class - Methods

Prototype: **None**

Returns The method does not return value.

The Construct method creates a new CompilerVariables 236 property based on the SBCompileVars 235 data type.

Example:

```
None
```

See also:
Destruct 254

3.30.4.22 Destruct

Prototype: **None**

Returns The method does not return value.

The Destruct method frees and disposes of the CompilerVariables property as well as the SBCommandLine property.

Example:

```
None
```

See also:
Construct 253

3.30.5 Tutorial

SetupBuilder Class *(ITSetupBuilderClass)*

The SetupBuilder Class has 22 methods (functions) that perform a variety of tasks related to SetupBuilder or installs. You can easily include these in your software. The SetupBuilder class has two main functionalities.

First is to make it easy to copy files from a common install folder, such as "All Users\Application Data" to a local folder such as "User Name\Application Data" This is very important when installing on Windows Vista and Windows 7 because the installer normally cannot install into user folders as it requires administrator access, which changes the user account.

The second is to compile SetupBuilder projects. This can be useful if you want to automate compiling your SetupBuilder project.

The following methods are used to copy installed files:

- SetGlobalCSIDL 249/SetLocalCSIDL 250

  This sets the CSIDL location for the source and destination respectively when copying files

from a common (global) location to a user specific (local) location. Note that KnownFolderID is currently not supported by the Utilities.

- CopyTheFiles 245

  This method copies all files and folders from the source folder to the destination folder. A partial path (PathString 238) is used to determine the full source path, i.e. it is appended to the CSIDL folder for both the source and the destination. For example if the CSIDL is CSIDL_COMMON_APPDATA, and the partial PathString 238 is "Icetips Creative\Utilities" this would mean that the source folder is set to "C:\Documents and Settings\All Users\Application Data\Icetips Creative\Utilities" The same partial folder is appended to the destination CSIDL folder. The PathString 238 is also used to update a registry key under HKCU\Software - in this case it would be "HKCU\Software\Icetips Creative\Utilities" with a value called FilesCopied 237. CopyTheFiles 245 creates this key and value if it doesn't exist and sets FilesCopied 237 to True. This method is used by the FinishTheInstall 344 method which you would generally call before you access any files in your software to ensure that user data has been set up.

- SetPathString 250

  The PathString 238 contains the partial path that is used in CopyTheFiles 245 to determine source and destination folders as well as the registry key to update.

- FinishInstall 246

  This is a "do-it-all" method that takes the CSIDL values for the source and destination and the partial path string and copies the files and does all the work. This method is all you need to use to copy the data and set everything right. You would normally call this method before you do any file access in your program to ensure that all user data has been set up.

The following methods are used to compile SetupBuilder projects:

- CompileSBProject 243

  Use this method to compile a SetupBuilder project. If you set the DestinationFolder 236 property before you call this method the project will be compiled into that folder. If the folder does not exist it will be created. Note that this only works in SetupBuilder 6.5 build 2000 or later. In earlier builds this will be ignored and the default destination used, which is the location of the project file with a new folder created with the same name as the project file.

- CompileSBProject 243

  Use this method to compile a SetupBuilder project. If you set the DestinationFolder 236 property before you call this method the project will be compiled into that folder. If the folder does not exist it will be created. Note that this only works in SetupBuilder 6.5 build 2000 or later. In earlier builds this will be ignored and the default destination used, which is the location of the project file with a new folder created with the same name as the project file.

- AddCompilerVariable 242

  This method can be used to pass values for compiler variables to your SetupBuilder project. This is limited to the amount of data that can be passed via the command line to the SetupBuilder compiler. Under Vista this is around 2K but under XP this should be close to 8K. This allows you to pass for example version number to user in the compile process. Note that passing the compiler variables to the project does NOT update the project, it simply instructs the compiler to use the values instead of the values in the project.

The demo application (UtilDemo.app 514) that is installed into the "Clarion\3rdParty\Examples\ITUtilities" folder contains a procedure called TestSetupBuilderClass 514 that demonstrates how to use these methods to compile a SetupBuilder project as well as copy the files.

3.31 Shell Class

3.31.1 Overview Shell Class

The Shell class has various methods that interact with the operating system shell, such as to run programs, copy files, create folders etc.

```
ITShellClass              Class(ITStringClass),TYPE,Module('ITShellClass.clw'),Link('ITShellClas
ShowSetting 258               Long       !! 2009-05-07:  Used to determine Show in ITRun
EnvVars                     &tEnvQueue


APIErrorHandler 260           Procedure(String pCaption),VIRTUAL
AboutShell 258                Procedure(String pApp, Long pW=IT_NULL, <String pOther>, Long pIc
AssociateProgram 260          Procedure(String pProgramExe, String pFileExt, String pFileTypeNa
CopyFiles 261                 Procedure(String pSource, String pDestination, Byte pCopyFilesOnl
CreateDirectory 262           Procedure(String pDirectory),Long,PROC
GenEnvVariables             PROCEDURE (),LONG,PROC
GetAssociatedProg 264         Procedure(String pFileName),String
GetAssociatedVerb 264         Procedure(String pExtension, <String pVerb>),String
GetEnvVar 265                 Procedure(String pEnvVar), String
SetEnvVar 274                 Procedure(String pEnvVar,String pValue),LONG
ExpandEnvString 263           Procedure(String pSt),String
GetExeFromExtension 266       Procedure(String pExtension),String
GetSpecialFolder 267          Procedure(Long pFolderID),String
ITRun 269                     Procedure(String pCommandLine, Long pWait=0, <String pParameters>
ITRunWait 271                 Procedure(String pCommandLine, Long pWait=0, <String pParameters>
ITRunFile 270                 Procedure(String pCommandLine, Long pWait=0, <String pVerb>, <Str
ITShellExec 272               Procedure(String pFile, <String pOp>, <String pParam>, <String pD
OpenURL 273                   Procedure(String pURL),VIRTUAL              !! AB 2006-03-06
ShellExec 275                 Procedure(Long pW, String pOp, String pFile, <String pParam>, <St
ShellExecEx 275               Procedure(*IT_SHELLEXECUTEINFO pShellExecInfo),IT_BOOL
ShowFilePropertyWindow 276    Procedure(String pFileName,<String pTab>)
PathIsDir 273                 Procedure(String pPath),Byte
Construct 82                  Procedure
Destruct 82                   Procedure
                          End
```

3.31.2 Data Types Shell Class

The Shell class uses the following datatypes:

IT_SHELLEXECUTEINFO 257

3.31.2.1 IT_SHELLEXECUTEINFO Shell Class - Data Types

This structure is used in the ShellExecEx method for the ShellExecuteEx api call. For more information see SHELLEXECUTEINFO structure on MSDN.

```
IT_SHELLEXECUTEINFO  GROUP,TYPE
cbSize                 IT_DWORD      ! ULONG
fMask                  ULONG
hwnd                   IT_HWND       ! UNSIGNED
lpVerb                 Long
```

```
lpFile                  Long
lpParameters            Long
lpDirectory             Long
nShow                   IT_int          ! SIGNED
hInstApp                IT_HINSTANCE    ! UNSIGNED
lpIDList                IT_LPVOID       ! ULONG
lpClass                 Long
hkeyClass               IT_HKEY         ! UNSIGNED
dwHotKey                IT_DWORD        ! ULONG
DUMMYUNIONNAME          Long            ! Pointer to union...
hProcess                IT_HANDLE       ! UNSIGNED
                      END
```

3.31.2.2 tEnvQueue

Shell Class - Data Types

This queue is used in the GenEnvVariables method to store the retrieved environment variables and their values.

```
tEnvQueue               QUEUE,TYPE
VarName                   CSTRING(101)
VarValue                  CSTRING(8192)
                        END
```

3.31.3 Properties

Shell Class

3.31.3.1 EnvVars

Shell Class - Properties

The EnvVars property is used in the GenEnvVariables methods and is used to store the environment variables and their data.

```
EnvVars                 &tEnvQueue
```

3.31.3.2 ShowSetting

Shell Class - Properties

This can be set before a call to ITRun 269, ITRunWait 271 and ITRunFile 270 to indicate the show type to use. Possible values are:

```
IT_SW_HIDE              EQUATE(0)
IT_SW_SHOWNORMAL        EQUATE(1)
IT_SW_SHOWMINIMIZED     EQUATE(2)
IT_SW_SHOWMAXIMIZED     EQUATE(3)
IT_SW_SHOWNOACTIVATE    EQUATE(4)
IT_SW_SHOW              EQUATE(5)
IT_SW_MINIMIZE          EQUATE(6)
IT_SW_SHOWMINNOACTIVE   EQUATE(7)
IT_SW_SHOWNA            EQUATE(8)
IT_SW_RESTORE           EQUATE(9)
```

3.31.4 Methods

Shell Class

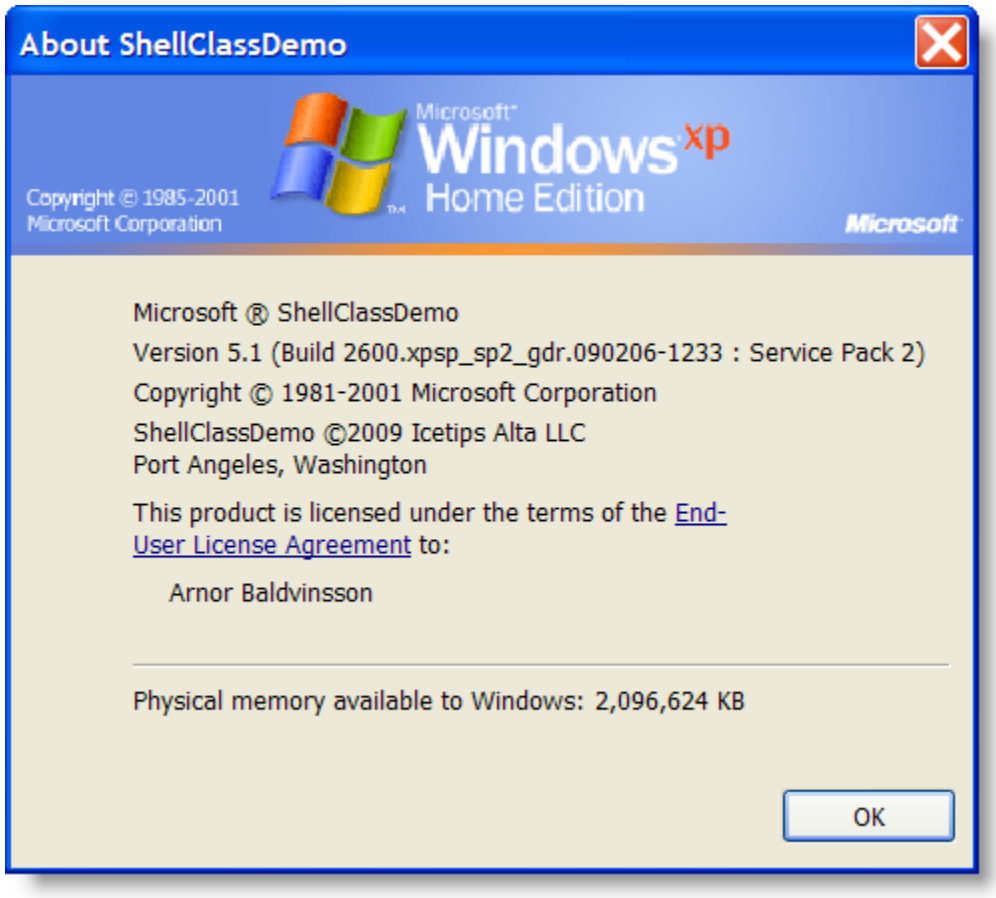
3.31.4.1 AboutShell

Shell Class - Methods

Prototype: **(String pApp, Long pW=IT_NULL, <String pOther>, Long pIcon=IT_NULL)**

pApp	Name of the app. This appears after the "Microsoft ® " in the first line on the window body.
pW	Handle of the parent window. This defaults to IT_NULL. Use PROP:Handle to get the handle of a window if you want to pass it as parameter.
pOther	String used below the Microsoft's copyright notice. This can be maximum two lines, separated with <13,10>
pIcon	Handle to an icon to use on the window.

The AboutShell method displays the standard "About" window. See below.



Example:

```
ITS  ITShellClass
 Code
 ITS.AboutShell('ShellClassDemo',,'ShellClassDemo ©2009 Icetips Alta LLC<13,10>Port
Angeles, Washington')
```

This code produced the window in the screenshot above.

3.31.4.2 APIErrorHandler

Prototype: **(String pCaption),VIRTUAL**

pCaption Caption for the message window

This method get's the last API error code and error text and shows a message window with the error information. This method uses the GetLastAPIError 66 method in the Core Class to get the last API error information.

Example:

```
ITS.APIErrorHandler('Error after copying files')
```

See also:
LastApiError 55
LastApiErrorCode 56
GetLastAPIError 66
GetLastAPIErrorCode 67

3.31.4.3 AssociateProgram

Prototype: **(String pProgramExe, String pFileExt, String pFileTypeName, Long pIconIndex=0),BYTE !! Returns True if association was successfull.**

pProgramExe Path and name of the executable to associate with the file extension specified in pFileExt

pFileExt Full file extension (i.e. in the format of ".extension" note the prefixed period) of the file type to associate with the executable file specified in pProgramExe

pFileTypeName Name for the file type.

pIconIndex Index for the icon to use with the association. Defaults to the default icon of the executable (0)

Returns Returns true if the association was successful.

This methods makes it easy to associate specific program with specific file extension so that double clicking on the file in Windows Explorer will load the program. Note that it is up to the developer to parse the command line parameters of the program.

Note that executing this method from a non-elevated program under Vista, Windows Server 2008 or Windows 7 will trigger virtualization writing to the registry keys and the information may be written to the HKEY_CURRENT_USER instead of HKEY_CLASSES_ROOT.

Example:

```
 If Message('This will associate ".abcdefg" with the ShellClassDemo program.  Do
you want to continue?',|
            'Associate extension with this
program?',ICON:Question,BUTTON:Yes+BUTTON:No,BUTTON:No) = BUTTON:Yes
   ITS.AssociateProgram(ITS.EXEName,'.abcdefg','Icetips.CoreClassDemo')
 End
```

See also:

GetAssociatedProg 264
GetAssociatedVerb 264
ShellExec 275
ShellExecEx 275
ITRunFile 270
OpenURL 273

3.31.4.4 CopyFiles

Shell Class - Methods

Prototype:	**(String pSource, String pDestination, Byte pCopyFilesOnly=True, Byte pCopySubDirs=False, Byte pConfirmCreateDest=False, Byte pSimpleProgress=False,Long pExtraAttrib=0, Byte pReplaceAttrib=0),Long,Proc**
pSource	The source folder and files. This can be a NULL character separated string with a list of fully qualified filenames.
pDestination	Destination folder or file.
pCopyFilesOnly	Set to True if you want to copy only files.
pCopySubDirs	Set to True if you want to copy subfolders as well.
pConfirmCreateDest	Set to True to confirm creation of the destination folder(s)
pSimpleProgress	Set to True if you want to display a simple progress window.
pExtraAttrib	Extra attributes/flags used in the fFlags field in the SHFILEOPSTRUCT
pReplaceAttrib	Can only be specified with pExtraAttrib. If it is set to true, the complete attribute set is replaced. If it is set to false, the pExtraAttrib value is added to the FileOp.fFlags property.
Returns	Returns 0 if the operation was successful. If the operation failed it returns a standard windows error code. See http://msdn.microsoft.com/en-us/library/bb762164.aspx for additional information on old error codes that this method can still return. To get the error code and formatted error message use the LastAPIError 55 or LastAPIErrorCode 56 class properties.

This method is designed to copy files from one place to another. Internally it uses SHFileOperation to accomplish that.

Please see http://msdn.microsoft.com/en-us/library/windows/desktop/bb759795(v=vs.85).aspx for information about what flags you can specify in the pExtraAttrib. Here are two examples on how to deal with a situation when you are copying files to a destination folder where the file exists

IT_FOF_RENAMEONCOLLISION will not prompt user but will create a new filename GLPOST - Copy.csv As for subsequent copy again, the filename will be GLPOST - Copy(2).csv

IT_FOF_NOCONFIRMATION will not prompt user and overwrite the existing file.

Example:

```
 If Message('Are you sure that you want to copy "' & Clip(Loc:Source) & '" to "' &
Clip(Loc:Destination) & '"?','Copy Files?',|
            ICON:Question,Button:Yes+Button:No,Button:No) = Button:Yes
```

```
   ! Copy the files
   ITS.CopyFiles(Loc:Source,Loc:Destination)
End
```

See also:
Windows System Error Codes on MSDN
SHFileOperation documentation on MSDN - including old error codes.

3.31.4.5 CreateDirectory

Prototype: **(String pDirectory),Long,PROC**

pDirectory Directory path to create

Returns Returns the return value for SHCreateDirectoryEx, see link to MSDN below. If the method succeeds it will return IT_ERROR_SUCCESS

This method can be used to create a single multi-level directory structure. If one or more path is not found it is created.

Example:

```
ITS  ITShellClass
 Code
 Loc:FolderToCreate = 'c:\test\test\test'
 ITS.CreateDirectory(Loc:FolderToCreate)
```

This will create the directory levels needed to construct this path.

See also:

http://msdn.microsoft.com/en-us/library/bb762131(VS.85).aspx

3.31.4.6 CreateFolder

Prototype: **(String pDirectory),Long,PROC**

pDirectory Directory path to create

Returns Returns the return value for SHCreateDirectoryEx, see link to MSDN below. If the method succeeds it will return IT_ERROR_SUCCESS

This method can be used to create a single multi-level directory structure. If one or more path is not found it is created. Note that this method simply calls CreateDirectory.

Example:

```
ITS  ITShellClass
 Code
 Loc:FolderToCreate = 'c:\test\test\test'
 ITS.CreateFolder(Loc:FolderToCreate)
```

This will create the directory levels needed to construct this path.

See also:

http://msdn.microsoft.com/en-us/library/bb762131(VS.85).aspx

3.31.4.7 ExpandEnvString

Prototype: **(String pSt, Byte pReturnEmpty=0),String**

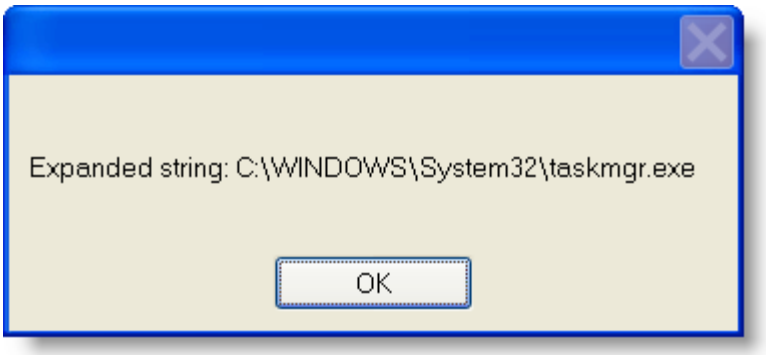
pSt	Environment string to expand
pReturnEmpty	Return an empty string if the environment variable is empty
Returns	The expanded environment string

This method is used to take a string that contains an environment variable, such as %PATH% and expand it. The environment variable can be the entire string or it can be just part of it. For example '%PATH%' or 'The path contains "%PATH%".'

Example:

```
ITS  ITShellClass
Loc:EnvStringToExpand String(255)
 Code
 Loc:EnvStringToExpand = '%SystemRoot%\System32\taskmgr.exe'
 Loc:EnvStringToExpand = ITS.ExpandEnvString(Loc:EnvStringToExpand)
 Message('Expanded string: ' & Loc:EnvStringToExpand)
```

This would result in a message similar to this:



See also:
SetEnvVar 274

3.31.4.8 GenEnvVariables

Prototype: **(),LONG,PROC**

Returns	Returns the number of environment variables retrieved from the operating system environment

Example:

```
ITS  ITStringClass
I    LONG
  CODE
  ITS.GenEnvVariables()
  LOOP I = 1 TO RECORDS(ITS.EnvVars)
    GET(ITS.EnvVars,I)
    ITS.ODS(FORMAT(I,@n_3) & ' ' & FORMAT(ITS.EnvVars.VarName,@s30) & ' =
' & ITS.EnvVars.VarValue)
  END
```

See also:
GetEnvVar 265

3.31.4.9 GetAssociatedProg

Prototype: **(String pFileName),String**

pFileName Name of the file to get associated program for.

Returns The full path and filename of the executable associated with the file. If no association is found, the method returns an empty string.

Use this method to find executable associated with specific file extension, such as .xls for MS Excel etc. Using any of the run or ShellExecute methods uses the same method to find the associated executable to be able to run files, open web pages etc. etc. Note that the file MUST exist. If the file does not exist use GetExeFromExtension 266.

Example:

```
ITS  ITShellClass
Loc:FileName  String(255)
 Code
 Loc:FileName = 'c:\test.aprj' !! Build Automator Extension
 Message('Program to open ' & Clip(Loc:FileName) & ': ' &|
         ITS.GetExeFromExtension(Loc:FileName),'Associated
Executable',ICON:Exclamation)
```

See also:
AssociateProgram 260
GetAssociatedVerb 264
GetExeFromExtension 266
ITRun 269
ITRunWait 271
ITShellExec 272
ShellExec 275
ShellExecEx 275

3.31.4.10 GetAssociatedVerb

Prototype: **(String pExtension, <String pVerb>),String**

pExtension Extension to find the verb information for

pVerb Verb to query. This is "open" by default.

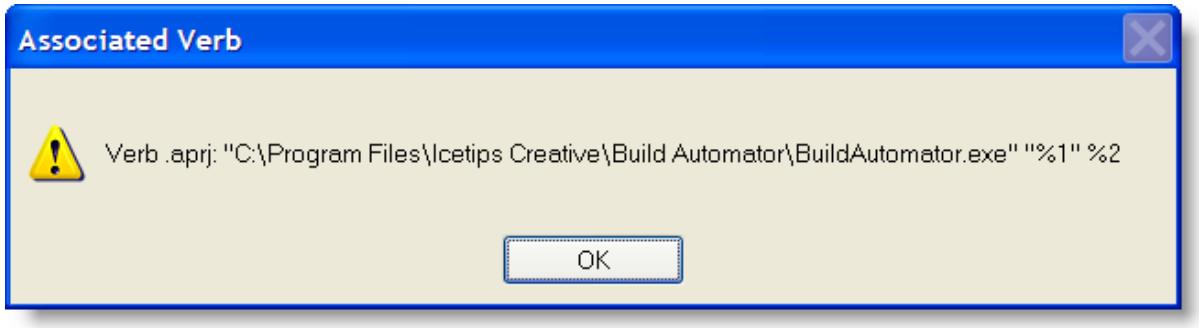
Returns The full verb information from the registry.

This is used to find the full command line used to run a program when a particular verb is used to execute the program with ShellExecute or related api calls.

Example:

```
ITS  ITShellClass
Loc:Extension  String(10)
 Code
 Loc:Extension = '.aprj' !! Build Automator extension
 Message('Verb ' & Clip(Loc:Extension) & ': ' &|
         ITS.GetAssociatedVerb(Loc:Extension),'Associated Verb',ICON:Exclamation)
```

This will result in a message something like this:



See also:
GetAssociatedProg 264
ITRun 269
ITRunWait 271
ITShellExec 272
ShellExec 275
ShellExecEx 275

3.31.4.11 GetEnvVar — Shell Class - Methods

Prototype: **(String pEnvVar), String**

pEnvVar The Environment variable to retrieve.

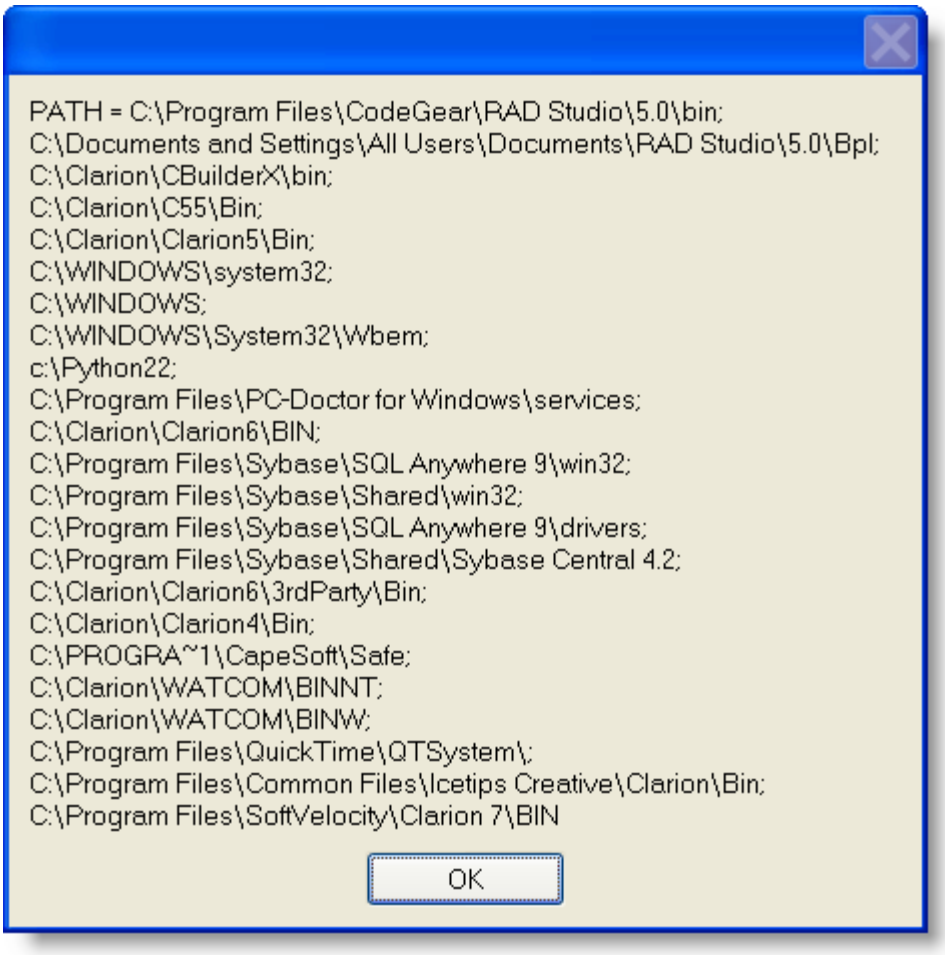
Returns String with the value of the environment variable

Use this method to get the value of a single environment variable.

Example:

```
ITS  ITShellClass
Loc:Path CString(10000)
 Code
 Loc:Path = ITS.GetEnvVar('PATH')
 ITS.SearchReplace(';',';<13,10>',Loc:Path)
 Message('PATH = ' & Loc:Path)
```

This results in a message like this one:



See also:
ExpandEnvString 263
SetEnvVar 274
http://msdn.microsoft.com/en-us/library/ms683188(VS.85).aspx

3.31.4.12 GetExeFromExtension

Prototype: **(String pExtension),String**

pExtension Extension to find executable for.

Returns The full path and filename of the associated executable if it is found. If not executable is found it returns an empty string.

This method can be used to find executable programs that are associated with extensions without the file existing. GetAssociatedProg requires that the file exists so GetExeFromExtension is an ideal replacement if the file doesn't exist. Note that the extension parameter must ONLY include the extension part, not a full filename.

Example:

```
ITS  ITShellClass
Loc:Extension  String(10)
 Code
 Loc:Extension = '.aprj' !! Build Automator Extension
 Message('Program to open ' & Clip(Loc:Extension) & ': ' &|
         ITS.GetExeFromExtension(Loc:Extension),'Associated
Executable',ICON:Exclamation)
```

See also:
GetAssociatedProg 264
GetAssociatedVerb 264

3.31.4.13 GetSpecialFolder

Shell Class - Methods

Prototype: **(Long pFolderID),String**

pFolderID One of the IT_CSIDL equates to is appropriate for the folder.

Returns String with the path or empty string if the CSIDL doesn't exist.

This method uses SHGetSpecialFolderPath API to retrieve the appropriate path string. Currently the Icetips Utilities do not support KNOWNFOLDERID. However, CSIDL are supported on Vista, Windows 2008 and Windows 7 so this method is fully valid on those operating systems. We expect to add support for KNOWNFOLDERID in early 2010.

Valid values for pFolderID are:

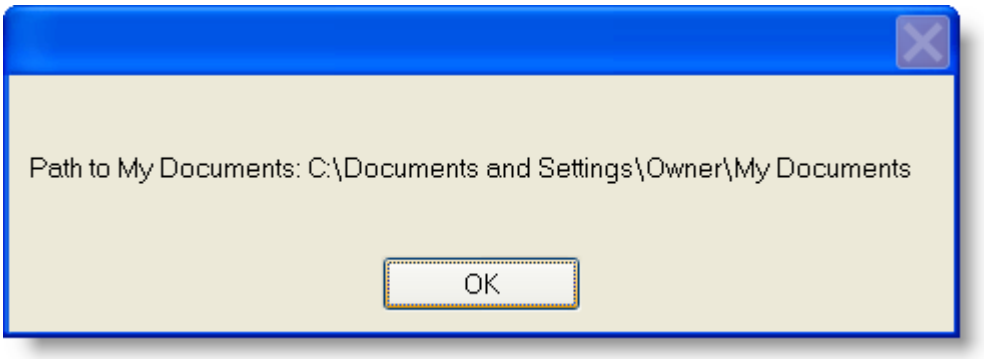
```
IT_CSIDL_DESKTOP
IT_CSIDL_INTERNET
IT_CSIDL_PROGRAMS
IT_CSIDL_CONTROLS
IT_CSIDL_PRINTERS
IT_CSIDL_PERSONAL
IT_CSIDL_FAVORITES
IT_CSIDL_STARTUP
IT_CSIDL_RECENT
IT_CSIDL_SENDTO
IT_CSIDL_BITBUCKET
IT_CSIDL_STARTMENU
IT_CSIDL_DESKTOPDIRECTORY
IT_CSIDL_DRIVES
IT_CSIDL_NETWORK
IT_CSIDL_NETHOOD
IT_CSIDL_FONTS
IT_CSIDL_TEMPLATES
IT_CSIDL_COMMON_STARTMENU
IT_CSIDL_COMMON_PROGRAMS
IT_CSIDL_COMMON_STARTUP
IT_CSIDL_COMMON_DESKTOPDIRECTORY
IT_CSIDL_APPDATA
IT_CSIDL_PRINTHOOD
IT_CSIDL_LOCAL_APPDATA
IT_CSIDL_ALTSTARTUP
IT_CSIDL_COMMON_ALTSTARTUP
IT_CSIDL_COMMON_FAVORITES
IT_CSIDL_INTERNET_CACHE
IT_CSIDL_COOKIES
IT_CSIDL_HISTORY
IT_CSIDL_COMMON_APPDATA
IT_CSIDL_WINDOWS
IT_CSIDL_SYSTEM
IT_CSIDL_PROGRAM_FILES
```

```
IT_CSIDL_MYPICTURES
IT_CSIDL_PROFILE
IT_CSIDL_SYSTEMX86
IT_CSIDL_PROGRAM_FILESX86
IT_CSIDL_PROGRAM_FILES_COMMON
IT_CSIDL_PROGRAM_FILES_COMMONX86
IT_CSIDL_COMMON_TEMPLATES
IT_CSIDL_COMMON_DOCUMENTS
IT_CSIDL_COMMON_ADMINTOOLS
IT_CSIDL_ADMINTOOLS
```

Example:

```
ITS  ITShellClass
 Code
 Message('Path to My Documents: ' & ITS.GetSpecialFolder(IT_CSIDL_PERSONAL))
```

This would result in a message like this:



See also:

http://msdn.microsoft.com/en-us/library/bb762494(VS.85).aspx

http://msdn.microsoft.com/en-us/library/bb762204(VS.85).aspx

3.31.4.14 IsUserAdmin

Prototype: **(),BYTE**

Returns True if the current user is logged in as administrator

This method can be used with programs running under User Access Control to test if the program is running with administrator credentials, i.e. if the program is running elevated.

Example:

```
ITS  ITShellClass
 Code
 If ITS.IsUserAdmin()
   Message('Program is elevated')
 Else
   Message('Program is NOT elevated')
 End
```

See also:

IsProgramElevated 269

ITRun 269

3.31.4.15 IsProgramElevated — Shell Class - Methods

Prototype: **(),BYTE**

Returns True if the current user is logged in as administrator

This method can be used with programs running under User Access Control to test if the program is running with administrator credentials, i.e. if the program is running elevated. Note that this simply a wrapper for the IsUserAdmin 268 method.

Example:

```
ITS  ITShellClass
 Code
 If ITS.IsProgramElevated()
   Message('Program is elevated')
 Else
   Message('Program is NOT elevated')
 End
```

See also:
IsUserAdmin 268
ITRun 269

3.31.4.16 ITRun — Shell Class - Methods

Prototype: **(String pCommandLine, Long pWait=0, <String pParameters>,<String pStartupFolder>,Byte pElevate=0),IT_HANDLE,PROC**

pCommandline Path and name of program or file to execute. If the path is a long path (i.e. including spaces and long filename) we advise that you use double quotes around the command line. Note that this should **not** contain any parameters strings, use the pParameters parameter for that.

pWait True or false to indicate if the method should wait for the program to finish. If this parameter is false the method returns a handle to the started process. Default value is false (0)

pParameters Optional parameters to pass to the executed program. If a parameter contains a long filename or path we advise that you use double quotes around that parameter value.

pStartupFolder Optional startup folder. If this parameter is supplied it must point to a valid folder and the program will then behave as it if was run in the Startup Folder.

pElevate Optional parameter that is only valid under Vista, Windows Server 2008, Windows 7 or later operating systems where User Account Control is active. For other operating systems or if UAC is turned off, this parameter is ignored. If set to True it will force the called program to run elevated.

Returns Returns the handle to the process if pWait is false. If pWait is true it returns the exit code from the called process or if no exit code was received it returns the return value from WaitForSingleObject API. Note: November 12, 2011: This has changed slightly, so that if the called process returns no exit code, the

method returns 0 instead of the return value from WaitForSingleObject. The reason for this change is that it is possible that if the called process returns nothing and WaitForSingleObject returns a value that the calling code is expecting, the results are wrong.

The ITRun method is a replacement for the RUN statement in the Clarion runtime library. The advantages of using ITRun is that it will automatically elevate a program if it needs to where as RUN() cannot do that. You can also set the startup folder which you cannot do with RUN and you can also force elevation under Vista and newer operating systems where User Account Control (UAC) is active.

Example:

```
ITS  ITShellClass
Loc:ProgramToRun  CString(2049)
Loc:WaitForProgram Byte
 Code
  Loc:WaitForProgram = True
  Loc:ProgramToRun   = '"C:\program files\Icetips Alta LLC\My program.exe"'
  If Loc:ProgramToRun
    ITS.ITRun(Loc:ProgramToRun,Loc:WaitForProgram)
  End
```

See also:

ITRunFile 270

ITRunWait 271

OpenURL 273

ShellExec 275

ShellExecEx 275

3.31.4.17 ITRunFile

Shell Class - Methods

Prototype:	**(String pCommandLine, Long pWait=0, <String pVerb>, <String pParameters>),IT_HANDLE,PROC**
pCommandline	Path and name of file to execute. If the path is a long path (i.e. including spaces and long filename) we advise that you use double quotes around the command line.
pWait	True or false to indicate if the method should wait for the program to finish. If this parameter is false the method returns a handle to the started process. Default value is false (0)
pVerb	The verb to use, such as IT_SE_Open, IT_SE_Print, IT_SE_Explore
pParameters	Optional parameters to pass to the executed program. If a parameter contains a long filename or path we advise that you use double quotes around that parameter value.
Returns	Returns the handle to the process if pWait is false. If pWait is true it returns the exit code from the called process or if no exit code was received it returns the return value from WaitForSingleObject API.

This method is very similar to the ITRun, except this method is more geared toward running files rather than programs. It does not have an option to elevate or set the folder path, but it allows you to set the

verb to use instead.

Available verbs are defined in the ITWin32Equates.inc file:

```
IT_SE_Open                EQUATE('open')
IT_SE_Print               EQUATE('print')
IT_SE_PrintTo             EQUATE('printto')
IT_SE_Explore             EQUATE('explore')
IT_SE_RunAs               EQUATE('runas')
IT_SE_Edit                EQUATE('edit')
IT_SE_Find                EQUATE('find')
IT_SE_Properties          EQUATE('properties')
```

Example:

```
ITS  ITShellClass
Loc:FileToRun      CString(2049)
Loc:WaitForProgram Byte
 Code
 Loc:WaitForProgram = True
 Loc:FileToRun      = '"C:\program files\Icetips Alta LLC\My spreadsheet.xls"'
 ITS.ITRunFile(Loc:FileToRun,Loc:WaitForProgram,False,IT_SE_OPEN)
```

See also:

ITRun 269

ITRunWait 271

OpenURL 273

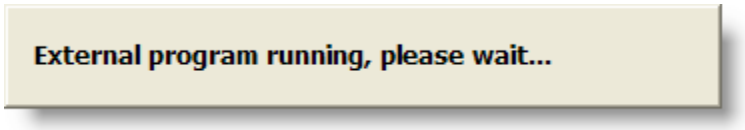
ShellExec 275

ShellExecEx 275

3.31.4.18 ITRunWait

Shell Class - Methods

Prototype:	**(String pCommandLine, Long pWait=0, <String pParameters>,<String pStartupFolder>,Byte pElevate=0, Byte pShowWaitWindow=1),IT_HANDLE,PROC**
pCommandline	Path and name of program or file to execute. If the path is a long path (i.e. including spaces and long filename) we advise that you use double quotes around the command line. Note that this should **not** contain any parameters strings, use the pParameters parameter for that.
pWait	True or false to indicate if the method should wait for the program to finish. If this parameter is false the method returns a handle to the started process. If this parameter is false, the pShowWaitWindow parameter is ignored and the wait window is not shown. Default value is false (0)
pParameters	Optional parameters to pass to the executed program. If a parameter contains a long filename or path we advise that you use double quotes around that parameter value.
pStartupFolder	Optional startup folder. If this parameter is supplied it must point to a valid folder and the program will then behave as it if was run in the Startup Folder.
pElevate	Optional parameter that is only valid under Vista, Windows Server 2008, Windows 7 or later operating systems where User Account Control is active. If set to True it will force the called program to run elevated.
pShowWaitWindow	Optional parameter to indicate if the Wait window should be shown or not:



This parameter defaults to True - i.e. to show the window.

Returns Returns the handle to the process if pWait is false. If pWait is true it returns the exit code from the called process or if no exit code was received it returns the return value from WaitForSingleObject API.

The ITRunWait method is a replacement for the RUN statement in the Clarion runtime library. The advantages of using ITRun is that it will automatically elevate a program if it needs to where as RUN() cannot do that. You can also set the startup folder which you cannot do with RUN and you can also force elevation under Vista and newer operating systems where User Account Control (UAC) is active.

Example:

```
ITS  ITShellClass
Loc:ProgramToRun  CString(2049)
Loc:WaitForProgram Byte
 Code
 Loc:WaitForProgram = True
 Loc:ProgramToRun   = '"C:\program files\Icetips Alta LLC\My program.exe"'
 If Loc:ProgramToRun
   ITS.ITRun(Loc:ProgramToRun,Loc:WaitForProgram)
 End
```

See also:
ITRun 269
ITRunFile 270
OpenURL 273
ShellExec 275
ShellExecEx 275

3.31.4.19 ITShellExec

Shell Class - Methods

Prototype: **(String pFile, <String pOp>, <String pParam>, <String pDir>, Long pShow=IT_SW_SHOWNORMAL),Long,PROC**

pFile Filename to execute.
pOp Optional verb to use.
pParam Optional parameters to pass.
pDir Optional default directory/folder to use.
pShow Optional Show parameters.

Returns If the file passed in pFile is not found the return value is -999. If it is found the return value is the return value from the ShellExecute api.

This is the same as the ShellExec 275 method except that the first parameter (window handle) is

skipped. For more information about the ShellExecute api I suggest reading about it on the Microsoft Developer Network (MSDN).

This method is a very powerful wrapper for the API and gives you all the power that API can give you.

Example:
```
ITS  ITShellExec
 Code
 ITS.ITShellExec('http://www.icetips.com')  !! Opens the Icetips website
 ITS.ITShellExec('C:\mydoc.doc',IT_SE_PRINT) !! Sends C:\mydoc.doc to the printer.
 ITS.ITShellExec('C:\Mybat.bat',IT_SE_OPEN,,,IT_SW_SHOWMINIMIZED)  !! Runs the .bat
file in a minimized window
```

See also:
ShellExec 275
ShellExecEx 275
ITRun 269
ITRunFile 270
ITRunWait 271

3.31.4.20 OpenURL — Shell Class - Methods

Prototype: **(String pURL),VIRTUAL**

pURL The URL to open. This can be any URL, http://, ftp:// etc.

This method is a wrapper for the ShellExec 275 method and simply calls it with the name of the url to open. You could also use this to open a file locally, but since it only takes the url or filename the ITRunFile 270 or ITShellExec 272 might be more appriate for file operations. This is a very simple method to use for quick opening of web sites or ftp sites.

Example:
```
ITS  ITShellClass
 Code
 ITS.OpenURL('http://www.icetips.com')  !! Open the Icetips webpage
```

See also:
ITRunFile 270
ITShellExec 272
ShellExec 275
ShellExecEx 275

3.31.4.21 PathIsDir — Shell Class - Methods

Prototype: **(String pPath),Byte**

pPath Path to check for validity.

Returns Returns true if the path is valid and exists, false if it does not exist or is not a path

This method can be used to determine if a filename being checked is a valid filename or a pathname. Note however that for it to detect if it is a path or not the file or path must exist. If the path or file does not exist, the method will return false even if the pathname/filename is valid.

In modern operating systems it is perfectly legal to have a file that does not have an extension. For example:

C:\test\this

A problem arises if you need to determine if a specified string is a valid filename or a valid folder name. Is C:\test\this a folder or is it a file without extension? PathIsDir will determine that based on the attributes of the filename. It uses PathIsDirectoryA API to detemine the validity of the name as a path. If the name is a folder name the method returns true, otherwise it returns false. Note that **the folder must exist** for it to return true.

Note: This API is loaded from shell32.dll. This API may not exist on Windows 95 and Windows 98 machines without Internet Explorer 4.0. In that case it will simply return False when called.

Example:

```
ITS  ITShellClass
 Code
 If ITS.PathIsDir('c:\documents and settings')
   Message ('this path is a folder')
 Else
   Message ('this path is not a folder')
 End
```

See also:

CreateDirectory 262

CreateFolder 262

GetSpecialFolder 267

3.31.4.22 SetEnvVar

Prototype: **(String pEnvVar,String pValue),LONG**

pEnvVar The name of the environment variable to set.

pValue The value to assign to the environment variable.

Returns False if the function failed. Other values indicate success.

This method is used to set the value of an environment variable. If the environment variable does not exist it is added to the environment.

Example:

```
ITS  ITShellClass
Env  String(20000)
```

```
 Code
 Env = ITS.GetEnvVar('PATH')     !! Get current PATH environment variable
 Env = Clip(Env) & ';' & Path()  !! Append ; + current path
 ITS.SetEnvVar('PATH',Env)       !! Set the PATH variable with the updated
information.
```

See also:
GetEnvVar 265

3.31.4.23 ShellExec — Shell Class - Methods

Prototype: **(Long pW, String pOp, String pFile, <String pParam>, <String pDir>, Long pShow=IT_SW_SHOWNORMAL),Long,PROC**

pW	Window handle to use.
pOp	Operator or verb to use.
pFile	Filename to execute.
pParam	Optional parameters
pDir	Optional startup directory
pShow	Optional show parameters.

Returns Returns the returnvalue from ShellExec api.

This method is a direct wrapper for the ShellExec api. The only difference is that it accepts string rather than CStrings and the 3 last parameters are optional.

Example:
```
ITS  ITShellClass
 Code
 ITS.ShelLExec(0{Prop:Handle},'C:\mydocs\letter.doc')  !! Opens a MS Word document
```

See also:
ITShellExec 272
ShellExecEx 275

3.31.4.24 ShellExecEx — Shell Class - Methods

Prototype: **(*IT_SHELLEXECUTEINFO pShellExecInfo),IT_BOOL**

pShelLExecInfo Pointer to a IT_SHELLEXECUTEINFO structure.

Returns Returns True if successful, false otherwise.

This method is a wrapper for the ShellExecuteEx API function.

See also:
ShellExec 275
ITShellExec 272

3.31.4.25 ShowFilePropertyWindow

Prototype: **(String pFileName,<String pTab>)**

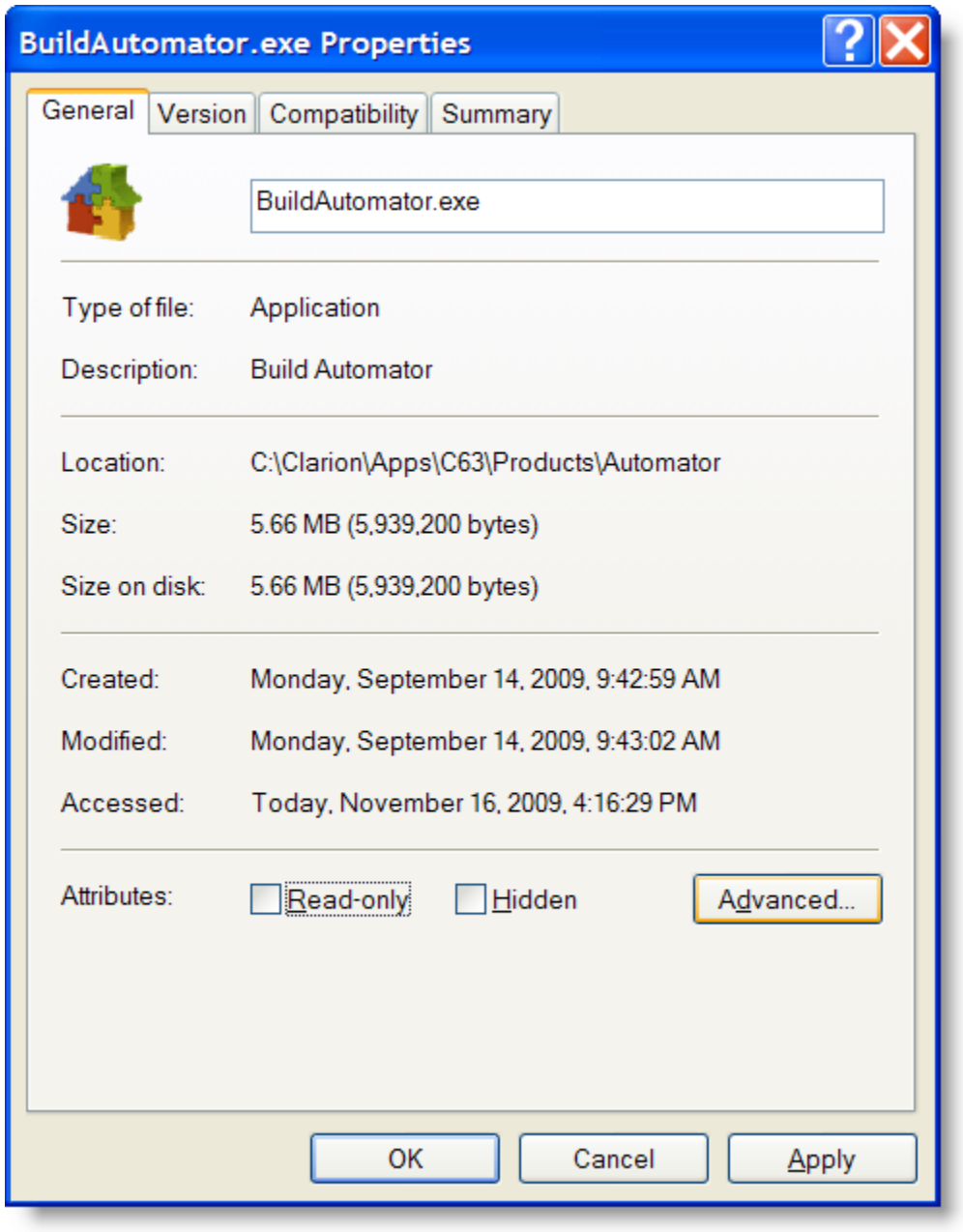
pFileName	Filename to show properties for
pTab	Optional name of the tab to show.

This method shows the "File Properties" window for the pFileName, optionally showing a specified tab. To select a specific tab you simply use the caption of the tab, i.e. "Compatibility" or "Version"

Example:

```
ITS  ITShellClass
 Code
 ITS.ShowFilePropertyWindow('C:\Program Files\Icetips
Creative\BuildAutomator\BuildAutomator.exe')
```

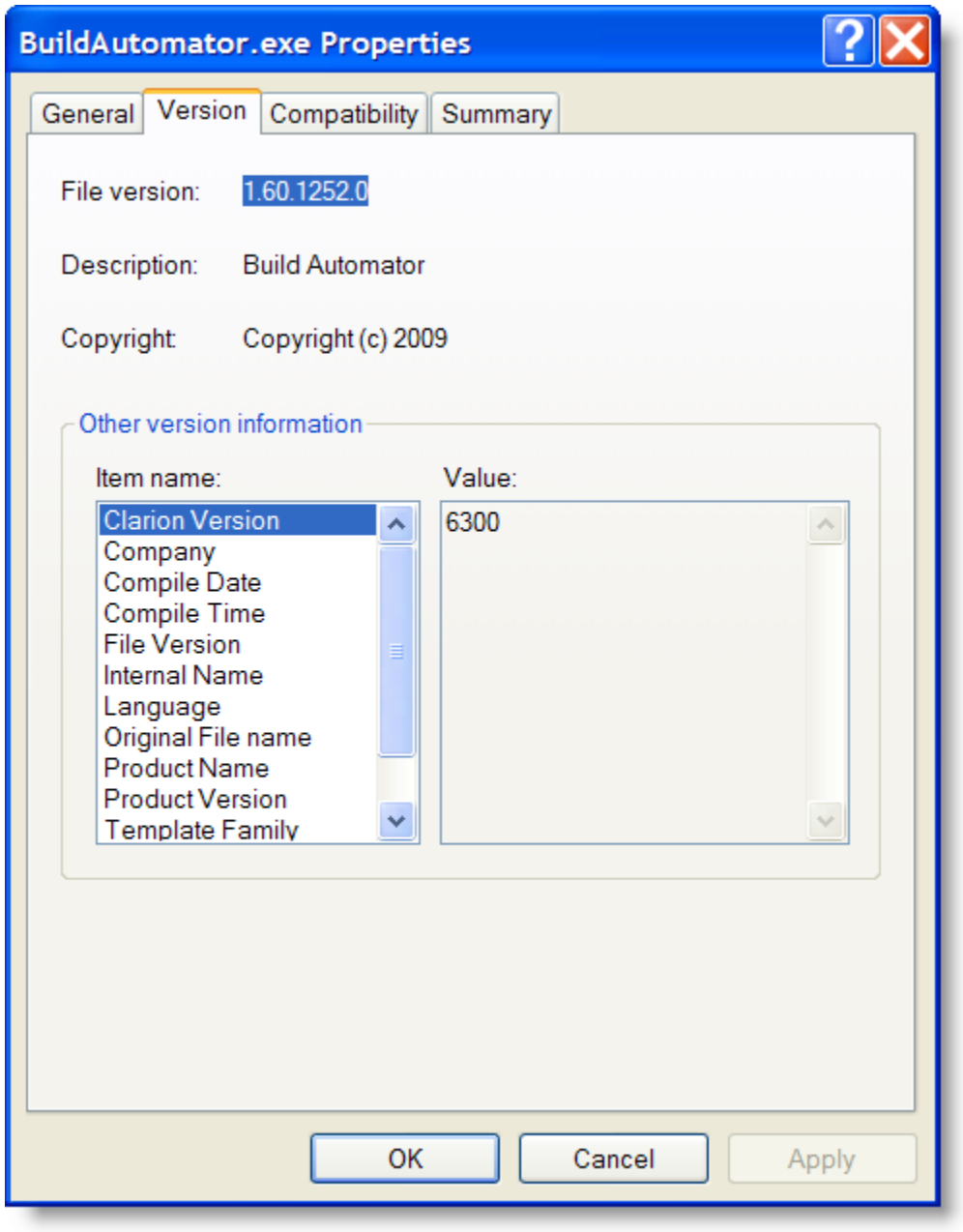
Will show this window:



Where as:

```
 ITS.ShowFilePropertyWindow('C:\Program Files\Icetips
Creative\BuildAutomator\BuildAutomator.exe','Version')
```

Will show this window:



3.31.4.26 Construct

Prototype: **None**

The Constructor only sets SELF.ShowSetting to IT_SW_SHOWNORMAL for default operation in the ITRun 269 method.

3.31.4.27 Destruct

Prototype: **None**

The Destructor does not have any code in it at this point.

3.32 String Class

3.32.1 Overview String Class

The string class has some very powerful string methods, including methods to read an entire file into a string buffer and write a string buffer to a file. It can also parse a string into lines or into words.

Check out the short tutorial 318 at the end of this chapter.

```
ITStringClass
Class(ITUtilityClass),TYPE,Module('ITStringClass.clw'),Link('ITStringClass'
,_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
BufferSize 283                  Long
FileString 284                  &String
Lines 284                       &ITLinesQ
CSVFields 284                   &ITLinesQ
ResStr 286                      &String,PRIVATE
TempS 286                       &String,PRIVATE
Words 286                       &ITWordQ 282
Found 284                       &tITFoundQ 282
WordCounter 286                 Long
HTMLString 284                  &String
SplitStringProgressFEQ 285      Long
DepunctuationString 283         String(255)
TreatEmptyLastItemAsLine 285    Byte
SkipEOLOnLastLine 285           BYTE
StringBetween 285               &STRING,PROTECTED
SetStringEnd 284                LONG
SetStringFound 285              LONG

AddIntoParentheses 287          PROCEDURE (String pOriginal, String
pAddition, <*CString pSeparator>),String
AddLine 288                     PROCEDURE (String pLine, Byte pNew=False)
AllocateFileString 289          PROCEDURE (Long pBytesToAllocate)  ! PRIVATE
AllocateLineString          PROCEDURE (Long pBytesToAllocate) ! PRIVATE
AllocateStringBetween       PROCEDURE (Long pBytesToAllocate) ! PRIVATE
AllocateXMLString           PROCEDURE (LONG pBytesToAllocate) ! PRIVATE
AppendToLine 289                PROCEDURE (String pLine)
CombineFieldName 290            PROCEDURE (String pFieldName, String
pPrefix,<String pTableName>),String
CompactString 291               PROCEDURE (String pOriginal, Byte
pUpperCase=False),String
CompareAndExtract 291           PROCEDURE (String pOriginal, String
pSearchFor),String
DebugLines 292                  PROCEDURE
DepunctuateString 292           PROCEDURE (*String pS, Byte
pAllowDigits=False)
DisposeFileString 293           PROCEDURE
DisposeLineString           PROCEDURE
DumpLinesInQ 293                PROCEDURE (Queue pQ, *? pLineField, <*?
pLineLenField>)
FileToLines 294                 PROCEDURE (String pFileName, <String
pDel>),Long,PROC
FileToString 295                PROCEDURE (String pFileName),String
```

```
FindInString 295                PROCEDURE (String pNeedle, String pHaystack,
Byte pCaseSensitive=False, Byte pSave=True),Long
FreeLines 298                   PROCEDURE
FreeString 298                  PROCEDURE (Byte pWords=0)
GetFieldPrefix 299              PROCEDURE (String pFieldName),String
GetLine 299                     PROCEDURE (Long pIndex),String
GetStringBetween 300            PROCEDURE (STRING pBegin, STRING pEnd, STRING
pSearchString,<Long pBeginPos>, <Long pEndBeginPos>, BYTE
pCaseSensitive=FALSE),STRING
GetWord 301                     PROCEDURE (Long pIndex),String
InsertString 301                PROCEDURE (STRING pStringToAdd, STRING
pStringToAddTo, LONG pPosition),STRING
LinesToFile 302                 PROCEDURE (String pFileName, <String
pEOL>),LONG,PROC
LinesToString 303               PROCEDURE (String pEOL),LONG
MatchParenthesis 303            PROCEDURE (String pS),Short
PadString 306                   PROCEDURE (String pStr, String pPad, Short
pLen, Byte pStart=0),String
ParseDelimtedLine 304           PROCEDURE (String pLine, String pDelimiter,
Byte pRemoveQuotes=True,<String pQuote>),Long,PROC
ParseCSVLine 305                PROCEDURE (STRING pLine, Byte
pStringsAreQuoted=TRUE,<String pDelimiter>, <STRING pQuoteChar>),LONG
ReadFileToQ 306                 PROCEDURE (String pFileName),Long,PROC
ReadFileToString 307            PROCEDURE (String pFileName),Long,PROC
RemoveHTML 308                  PROCEDURE (String pHTMLString, Byte
pForceSpace=True),String
SetDepunctuationString 308      PROCEDURE (String pDepunct)
SetLineValue 308                PROCEDURE (String pLine)
SetSplitStringProgress 310      PROCEDURE (Long pProgressFEQ)
SetStringBetween 309            PROCEDURE (STRING pBegin, STRING pEnd, STRING
pSearchString, STRING pInsertString, BYTE pHandling, <Long pBeginPos>,
<Long pEndBeginPos>, BYTE pCaseSensitive=FALSE),STRING
SplitFieldName 310              PROCEDURE (String pFieldName, <*String
pPrefix>),String
SplitString 311                 PROCEDURE (String pStr,String
pDelimiter),Long,PROC
StringFromLines 312             PROCEDURE (),STRING
StringToFile 312                PROCEDURE (String pFileName, String pContent,
Byte pAppend=False),Long,PROC
StringToLines 313               PROCEDURE (String pS, <String
pDel>),Long,PROC
StringToWords 314               PROCEDURE (String pS, Byte pCount=True, Byte
pCaseSensitive=False),Long,PROC
StripParenthesis 315            PROCEDURE (String pTxt, <String
pParLeft>,<String pParRight>),String
UseEither 315                   PROCEDURE (String pS1, String pS2, Byte
pFavourite=1),String
WriteQToFile 316                PROCEDURE (String pFileName)
WriteStringToFile 316           PROCEDURE (String pFileName, String pContent,
Byte pAppend=False),Long,PROC

Construct 317                   PROCEDURE
Destruct 317                    PROCEDURE
                          End
```

3.32.2 Data Types String Class

The String class uses two datatypes, both queues, that are used to store parsed lines and words.

ITLinesQ 282
ITWordQ 282

3.32.2.1 ITWordQ String Class - Data Types

The ITWordQ is used for the Words 286 property for the StringToWords 314 method that splits a string into words.

```
ITWordQ                 QUEUE,TYPE
Word                      CString(61)
Len                       Byte
Counter                   Long
                        END
```

See also:
StringToWords 314
Words 286

3.32.2.2 ITLinesQ String Class - Data Types

The ITLinesQ is used for the Lines property which is used by several methods. Note that this structure was modified on March 30, 2009 to allow dynamic allocation for the string rather than fix it to 1K.

```
ITLinesQ                QUEUE,TYPE
OL                        &CString !! CString(1025)
Len                       Long
                        END
```

See also:
Lines 284

3.32.2.3 tITFoundQ String Class - Data Types

The iITFoundQ is declared in ITEquates.inc.

```
tITFoundQ               QUEUE,TYPE
StartPos                  Long
EndPos                    Long
                        END
```

See also:
Found 284
FindInString 295

3.32.3 Equates String Class

3.32.3.1 ITStIns:Prefix String Class - Equates

This equate is used with SetStringBetween in the pHandling parameter to indicate that the string should be added right after the pBegin string.

3.32.3.2 ITStIns:Replace String Class - Equates

This equate is used with SetStringBetween in the pHandling parameter to indicate that the string between pBegin and pEnd should be replaced with the pInsertString passed in to the method.

3.32.3.3 ITStIns:Append String Class - Equates

This equate is used with SetStringBetween in the pHandling parameter to indicate that the pInsertString should be added right before the pEnd string.

3.32.4 Properties String Class

The String Class has 10 public properties and 2 private properties.

```
BufferSize 283          Long
FileString 284          &String
Lines 284               &ITLinesQ
ResStr 286              &String,PRIVATE
TempS 286               &String,PRIVATE
Words 286               &ITWordQ
Found 284               &tITFoundQ
WordCounter 286         Long
HTMLString 284          &String
SplitStringProgressFEQ 285  Long
DepunctuationString 283 String(255)
TreatEmptyLastItemAsLine 285  Byte
```

3.32.4.1 BufferSize String Class - Properties

This property is used by the AddLine 288 and WriteQToFile 316 methods. It is increased in the AddLine 288 method so that when the WriteQToFile 316 is called the buffer size is known and the entire buffer can be allocated.

3.32.4.2 DepunctuationString String Class - Properties

Set by the SetDepunctuationString 308 method and used in the DepunctuateString 292 method.

3.32.4.3 LineString String Class - Properties

The LineString property is used in the LinesToString methods and is used to dynamically store the entier string in the Lines queue property.

It is declared as:

```
LineString                  &String
```

3.32.4.4 FileString

FileString is a dynamic string variable that is used by the ReadFileToString 307, WriteStringToFile 316 and FileToString 295 methods. It is used to dynamically create a buffer to contain the entire contents of a file when it's read or before it is written. This makes it extremely fast to read or write big chunks of text from or to a file.

3.32.4.5 Found

The Found property is used in the <methods used in> methods and is used to ...

It is declared as:

```
Found                       &tITFoundQ 282
```

3.32.4.6 HTMLString

This property is used by the RemoveHTML 308 method and is dynamically allocated when needed.

3.32.4.7 CSVFields

The CSVFields property is used in the ParseCSVLine methods and is used to store the data from each field.

It is declared as:

```
CSVFields                   &ITLinesQ
```

3.32.4.8 LineCnt

The LineCnt property is used in the AddLine and SplitString methods and is used to store the number of lines in the Lines queue property.

It is declared as:

```
LineCnt                     LONG
```

3.32.4.9 Lines

Lines is a queue of the ITLinesQ 282 type. It is used to store text, one line at the time, in a queue. This queue can then be looped through and processed, or written to a file with WriteQToFile 316.

3.32.4.10 SetStringEnd

The SetStringEnd property is used in the SetStringBetween 309 method. It contains the end position of the pEnd parameter in the resulting string (StringBetween 285) after the string has been reconstructed in SetStringBetween 309.

It is declared as:

```
SetStringEnd                LONG  !! AB 2015-10-29:  Specifies the end of
the closing pEnd string in SetStringBetween.
```

3.32.4.11 SetStringFound

The SetStringFound property is used in the SetStringBetween 309 method. It contains the starting location of pBegin in the resulting string (StringBetween 285) after the string has been reconstructed in SetStringBetween 309. If this property is 0 after a call to SetStringBetween there are no more pairs of pBegin/pEnd in the resulting string. If it is not 0 then there are one or more pairs of pBegin/pEnd inside the string and SetStringBetween could or should be called again to insert a new string.

It is declared as:

```
SetStringFound              LONG  !! AB 2015-10-29:  Specifies the location
of pBegin in the resulting string.  0 indicates not found.
```

3.32.4.12 SkipEOLOnLastLine

The SkipEOLOnLastLine property is used in the WriteQToFile method and is used to indicate if the last line written to the file should have the End Of Line delimiter added to it or not. If it is FALSE, an extra empty line is added at the end of the file. If it is TRUE the last line does not have a line termination delimiter so there is no empty line. Default value is FALSE.

It is declared as:

```
SkipEOLOnLastLine           BYTE
```

3.32.4.13 SplitStringProgressFEQ

This property is used to optionally specify a Progress FEQ for the SplitString 311 Method.

3.32.4.14 StringBetween

The StringBetween property is used in the SetStringBetween 309 method. It contains the resulting string after the pInsertString has been properly inserted into the pSearchString. This string is returned from SetStringBetween so you normally do not need to access it. However if you do, you can. Note that it is dynamically allocated so you can check the size of it if you need to create a string to contain it.

It is declared as:

```
StringBetween               &STRING,PROTECTED
```

3.32.4.15 TreatEmptyLastItemAsLine

The **TreatEmptyLastItemAsLine** property is only used in the SplitString 311 method and is used to determine if an empty last item in the array/list should be treated as a line.

Note that the SplitString 311 method is called from StringToLines 313 which in turn is called from FileToLines 294 so you can now determine how SplitString 311 behaves for both those methods by using the TreatEmptyLastItemAsLine property.

Consider this string for example:

```
'Arnor|Baldvinsson|'
```

Should that last item be treated as an empty line or as a non-existing line? The answer can be yes - or no, depending on the way you need this to work! For some purposes this must be three lines, while for other it must be only two lines. With the TreatEmptyLastItemAsLine property you can determine if it should create an empty line or not.

This code:

```
ITS  ITStringClass
St   CString(1024)
 Code
 ITS.TreatEmptyLastItemAsLine = False
 St = 'Arnor|Baldvinsson|'
 ITS.SplitString(st,'|')
```

will show only **two lines,** while:

```
ITS  ITStringClass
St   CString(1024)
 Code
 ITS.TreatEmptyLastItemAsLine = True
 St = 'Arnor|Baldvinsson|'
 ITS.SplitString(st,'|')
```

will show **three lines**, where the last one is empty.

It is declared as:

```
TreatEmptyLastItemAsLine   Byte
```

Added on June 22, 2012

3.32.4.16 WordCounter — String Class - Properties

This property is used by the StringToWords 314 method to set the total number of words in the string to.

3.32.4.17 Words — String Class - Properties

Word is a queue of the ITWordQ 282 type. It is used in the StringToWords 314 method which splits a string into words.

3.32.4.18 Private Properties — String Class - Properties

3.32.4.18.1 ResStr

This is a dynamic string variable used in the AddIntoParenthesis 287 method.

3.32.4.18.2 TempS

This is a dynamic string variable used for temporary storage in the AddIntoParenthesis 287 method.

3.32.5 Methods String Class

The String Class has 27 methods including the constructor and destructor.

```
AddIntoParentheses 287          Procedure(String pOriginal, String pAddition,
<*CString pSeparator>),String
AddLine 288                     Procedure(String pLine, Byte pNew=False)
AllocateFileString 289          Procedure(Long pBytesToAllocate)
AppendToLine 289                Procedure(String pLine)
CombineFieldName 290            Procedure(String pFieldName, String pPrefix,<String
pTableName>),String
CompactString 291               Procedure(String pOriginal, Byte
pUpperCase=False),String
CompareAndExtract 291           Procedure(String pOriginal, String pSearchFor),String
DebugLines 292                  Procedure
DepunctuateString 292           Procedure(*String pS, Byte pAllowDigits=False)
DisposeFileString 293           Procedure
DumpLinesInQ 293                Procedure(Queue pQ, *? pLineField, <*? pLineLenField>)
FileToLines 294                 Procedure(String pFileName),Long,PROC
FileToString 295                Procedure(String pFileName),String
FindInString 295                Procedure(String pNeedle, String pHaystack, Byte
pCaseSensitive=False, Byte pSave=True),Long
FreeLines 298                   Procedure
FreeString 298                  Procedure(Byte pWords=0)
GetFieldPrefix 299              Procedure(String pFieldName),String
GetLine 299                     Procedure(Long pIndex),String
GetWord 301                     Procedure(Long pIndex),String
MatchParenthesis 303            Procedure(String pS),Short
PadString 306                   Procedure(String pStr, String pPad, Short pLen, Byte
pStart=0),String
ReadFileToString 307            Procedure(String pFileName),Long,PROC
SetDepunctuationString 308      Procedure(String pDepunct)
SetLineValue 308                Procedure(String pLine)
SetSplitStringProgress 310      Procedure(Long pProgressFEQ)
SplitFieldName 310              Procedure(String pFieldName, <*String pPrefix>),String
SplitString 311                 Procedure(String pStr,String pDelimiter),Long,PROC
StringToLines 313               Procedure(String pS),Long,PROC
StringToWords 314               Procedure(String pS, Byte pCount=True, Byte
pCaseSensitive=False),Long,PROC
StripParenthesis 315            Procedure(String pTxt, <String pParLeft>,<String
pParRight>),String
UseEither 315                   Procedure(String pS1, String pS2, Byte
pFavourite=1),String
WriteQToFile 316                Procedure(String pFileName)
WriteStringToFile 316           Procedure(String pFileName, String pContent),Long,PROC

Construct 317                   Procedure
Destruct 317                    Procedure
```

3.32.5.1 AddIntoParenthesis String Class - Methods

Prototype: **(String pOriginal, String pAddition, <*CString pSeparator>),String**

pOriginal	Text to modify
pAddition	Text to add into parenthesis
pSeparator	Optional separator

Returns Returns modified text

This method can be used to place text into parenthesis.

Example:

```
ITS  ITStringClass
Orig  String(50)
Add   String(10)
Sep   String(2)
 Code
 Orig = 'This or (what)'
 Add  = 'ever'
 Sep  = ''
 Orig = ITS.AddIntoParenthesis(Orig,Add,Sep)
 ! Orig now contains 'This or (whatever)'
 Add = 'else'
 Sep = '-'
 Orig = ITS.AddIntoParenthesis(Orig,Add,Sep)
 ! Orig now contains 'This or (whatever-else)'
```

See also:
MatchParenthesis 303
StripParenthesis 315

3.32.5.2 AddLine

String Class - Methods

Prototype: **(String pLine, Byte pNew=False)**

pLine String to add to the Lines queue

pNew Indicates if the queue should be cleared. If True, the Lines queue is freed.

The AddLine method adds a single line of text to the Lines 284 queue property of the class. In version 1.1.2319 and earlier this was limited to 1024 characters. It is now dynamic and only limited by available memory) The text line is stored in the OL field of the queue and the length of the line is stored in the Len field of the queue. See the ITLinesQ 282 for the type declaration of the Lines 284 queue.

Example:

```
ITS  ITStringClass
 Code
 ITS.AddLine('This is a test',True)  !! First line, so free the queue
 ITS.AddLine('Second line')
```

At this point the ITS.Lines will contain two records.

See also:
Lines 284
WriteQToFile 316
AppendToLine 289
SetLineValue 308

3.32.5.3 AllocateFileString

Prototype: **(Long pBytesToAllocate)**

pBytesToAllocate Number of Bytes to allocate

This method is used to allocate memory for the FileString 284 property, which is used by the ReadFileToString 307, WriteStringToFile 316 and FileToString 295 methods. In normal operation you never need to call this method and should regard it as a PRIVATE method. However, **you can** also use this method if you need a quick dynamic string. Simply use AllocateFileString to create the string size you want and then use the FileString 284 property as you want. You can dispose of the string with DisposeFileString 293, but you don't need to call it if you want to repeatedly allocate the FileString 284 as the AllocateFileString 289 calls DisposeFileString 293 before it allocates it again, so there is no risk of memory leaks. The string is deallocated in the destructor 317 as well. Note that there is only one instance of the FileString active! So calling AllocateFileStrings again will dispose of the existing string and create a new one.

Example:

```
ITS  ITStringClass
 Code
 ITS.AllocateFileString(1024)
 ITS.FileString = 'This is a test'
 ITS.AllocateFileString(1024)
```

At this point the FileString is an empty 1K string as the second call disposes of the first string and creates a new one.

See also:
FileString 284

ReadFileToString 307

WriteStringToFile 316

FileToString 295

DisposeFileString 293

3.32.5.4 AppendToLine

Prototype: **(String pLine)**

pLine String to append to the currently selected line

This method is used to append text to the currently selected line, for example after a call to the GetLine 299 method. Note that no space is added between the existing data and the new data. If you want additional space make sure that you add it to the data that you pass to the AppendToLine 289 method.

Example:

```
ITS  ITStringClass
 Code
 ITS.AddLine('One',True)
 ITS.AddLine('Two')
```

```
ITS.AddLine('Three')
If ITS.GetLine(2)
  ITS.SetLineValue('Twenty')
  ITS.AppendToLine(' two')
End
```

Now the value of the second line is "Twenty two"

See also:
GetLine 299

AddLine 288

SetLineValue 308

3.32.5.5 CombineFieldName

String Class - Methods

Prototype: **(String pFieldName, String pPrefix,<String pTableName>),String**

pFieldName Name of field/column without prefix.
pPrefix Prefix name without trailing colon.
ptableName Optional name of the table for the Field/column.

Returns Combined field/column name.

This method can be used to combine a field or column name with either a prefix or a table name. This is handy when generating Clarion code where you know the table name or the prefix but have to parse out the fieldname and then later on combine them (perhaps with a different table name or prefix). See also the SplitFieldName 310 method which does the opposite of the CombineFieldName. If the table name is omitted the prefix and fieldname are combined as Prefix:FieldName, i.e. with a colon between them. If the table name is not omitted the table name is used instead of the prefix as TableName.FieldName, i.e. with a period between them. The case of the prefix, tablename and fieldname is not changed.

Example:

```
ITS  ITStringClass
Fn   String(20)
Pf   String(5)
Tn   String(20)
 Code
 Fn = 'SysID'
 Pf = 'MYF'
 Tn = 'MyFile'
 Message('Combined Fieldname with prefix: ' & ITS.CombineFieldName(Fn,Pf)
&|
         '|Combined Fieldname with tablename: ' & ITS.CombineFieldName(Fn,
Pf,Tn))
```

See also:
SplitFieldName 310

3.32.5.6 CompactString

Prototype: **(String pOriginal, Byte pUpperCase=False),String**

pOriginal String to compact
pUpperCase Indicates if the resulting string should be uppercased as well.

Returns The pOriginal string with all spaces and puncuation removed

This method removes all spaces and punctuation from a string. This can be very useful to create strings for sorting queues or tables where punctuation and spaces should not affect the sorting order. It is also very convenient when comparing strings where spaces and punctuation may differ, but the text is the same. This method calls the DepunctuateString 292 method to remove punctuation. The call to DepunctuateString allows numbers to be included so that only punctuation characters are removed. See also SetDepunctuationString 308 method and Construct 317 methods for more information about punctuation characters.

Example:

```
ITS  ITStringClass
S1   String(100)
S2   String(100)
 Code
 S1 = 'Arnor Baldvinsson, Icetips  Creative, Inc.'
 S2 = 'Arnor  Baldvinsson Icetips Creative. inc.'
 If ITS.CompactString(S1) = ITS.CompactString(S2)
   !! Text of strings match
 End
```

In this example the compact versions of the S1 and the S2 strings match, even though the spaces and punctuation makes the different.

See also:
DepunctuateString 292
SetDepunctuationString 308
Construct 317

3.32.5.7 CompareAndExtract

Prototype: **(String pOriginal, String pSearchFor),String**

pOriginal Original string to compare
pSearchFor String to search for

Returns Returns extracted string

This method can be used to extract string from another string.

Example:

```
ITS  ITStringClass
 Code
```

```
Message(ITS.CompareAndExtract('This or that','This or'))
```

This call would return ' that'

See also:
CompactString 291

3.32.5.8 DebugLines

This method has no parameters and does not return any information.

This method can be used to send the contents of the Lines queue to debug viewer such as DebugView to see exactly what is stored in the queue.

Example:

```
ITS  ITStringClass
 Code
 ITS.AddLine('First',True)
 ITS.AddLine('Second')
 ITS.AddLine('Third')
 ITS.DebugLines
```

This will show the 3 lines in DebugView

See also:
AddLine 288
Lines 284

3.32.5.9 DepunctuateString

Prototype:	**(*String pS, Byte pAllowDigits=False)**
pS	String to depunctuate.
pAllowDigits	Indicates that numeric digits are allowed in the string.
Returns	pS without any punctuation characters.

This method can be used to remove all punctuation from a string. This includes all control characters with ASCII value less than 32. The method uses the IsPunct function to determine if the character is punctuation or not. If pAllowsDigits is true, the the method also uses IsDigit to determine if the character is a digit or not. Note that this does not remove spaces. For that use the CompactString 291 method.

Note: On September 18, 2009 we added ()[]{}!@#$%^&*- as punctuation characters that are replaced with a space in this method. Note that this is in addition to the original usage of IsPunct etc. You can use the SetDepunctuationString method to change this.

Example:

```
ITS  ITStringClass
S1   String(100)
```

```
Code
S1 = 'Icetips Creative, Inc.'
ITS.DepunctuateString(S1)
! S1 would now be 'Icetips Creative Inc'
```

See also:
CompactString 291

SetDepunctuationString 308

3.32.5.10 DisposeFileString — String Class - Methods

This method has no parameters and does not return any information.

The DisposeFileString method is used to de-allocate memory used by the FileString 284 property, which is used by the ReadFileToString 307, WriteStringToFile 316 and FileToString 295 methods. In normal operation you **never** need to call this method and should regard it as a PRIVATE method. It is called automatically when you call the AllocateFileString method and also from the destructor of the class. Even if you use the AllocateFileString method to allocate dynamic string for you own you you do not need to call this method to deallocate the memory used as it is taken care of in both the AllocateFileString 289 and the Destruct 317 methods.

See also:
AllocateFileString 289
FileString 284

ReadFileToString 307

WriteStringToFile 316

FileToString 295

3.32.5.11 DumpLinesInQ — String Class - Methods

Prototype:	**(Queue pQ, *? pLineField, <*? pLineLenField>)**
pQ	The queue to put the fields in. This would normally be a local queue in a procedure
pLineField	The field to take the contents of the OL field
pLineLenField	The field to take the contents of the Len field.

This method loops through the Lines property queue and adds each record to the passed queue, using the reference to the line and length fields.

Example:

```
ITS  ITStringClass
Q    Queue
L      String(1024)
     End
  Code
  ITS.ReadFileToString('C:\temp\textfile.txt')
  ITS.SplitString(ITS.FileString,'<13,10>')
  ITS.DumpLinesInQ(Q,Q.L)
```

3.32.5.12 EncodeXML

Prototype: **(STRING pXML),STRING**

pXML The XML string to encode

Returns Returns the encoded XML string

This method takes an XML string and encodes characters > 127 as &#xxx; where xxx is the appropriate character number. It also encodes ampersand (&), less than "<", greater than ">", single quote ' and double quote " It can also be used to encode HTML.

Example:

```
ITS  ITStringClass
XMLStr  CSTRING(1025)
  CODE
  XMLStr = 'Arnór'
  XMLStr = ITS.EncodeXML(XMLStr)
  !! XMLStr now contains Arn&#243;r
```

See also:
FormatXML 296

3.32.5.13 FileToLines

Prototype: **(String pFileName, <String pDel>),Long,PROC**

pFileName Name of (text) file to read into lines

pDel Optional delimiter to use as End-Of-Line string. Added 2015-09-12

Returns Integer value containing the number of lines read into the Lines 284 property.

This method calls the FileToString 295 and then the StringToLines 313 methods to read and parse the file into a Lines 284 queue.

Example:

```
Fn   CString(2049)
I    Long
L    Long
ITST ITStringClass
 Code
 Fn = 'c:\temp\textfile.txt
 If Exists(ShortPath(Fn))
   L = ITST.FileToLines(Fn)
   Loop I = 1 To L
     ITST.ODS(Format(I,@n4) & ' ' & ITST.GetLine(I))
   End
 End
```

See also:
FileToString 295
StringToLines 313

3.32.5.14 FileToString — String Class - Methods

Prototype: **(String pFileName),String**

pFileName Name of (text) file to read into string

Returns String containing the contents of the file

This method calls the ReadFileToString 307 method and returns the contents of the FileString 284 property. After this method is called the FileString property is set to the entire contents of the pFileName file.

Example:
```
ITS  ITStringClass
S    String(1024)
  Code
  S = ITS.FileToString('test.txt')
  ! S now contains the contents of test.txt or the first 1024 bytes if
test.txt is larger than 1024 bytes.
```

See also:
FileString 284
ReadFileToString 307
WriteStringToFile 316
FileToLines 294

3.32.5.15 FindInString — String Class - Methods

Prototype: **(String pNeedle, String pHaystack, Byte pCaseSensitive=False, Byte pSave=True),Long**

pNeedle The string to search for.

pHaystack The string to search in.

pCaseSensitive Indicates if the search should be case sensitive or not. By default it is not case sensitive.

pSave Indicates if the resulting finds should be saved to the Found 284 queue property.

Returns The number of times the needle is found in the haystack.

This is a very powerful search method, similar to the InString function in Clarion - in fact it uses the InString() function internally. But instead of returning the start position like InString() does, it returns you the number of instances that the needle string is found in the haystack string. It can also save the start and end position of each found string (needle) in the search string (haystack) so it is easy to split

and splice the string. The results are stored in the Found 284 queue property.

Example:

```
ITS  ITStringClass
  Code
  Message('Found: ' & ITS.FindInString('test','This is a test to find
"Test" in the TestString'))
  !! Will show 3.  ITS.Found now has 3 records in it with the start/end
position of each "test" find.
```

See also:
Found

3.32.5.16 FormatXML

Prototype: **(STRING pXML, BYTE pIndentSpaces=2),STRING**

pXML XML string to format.

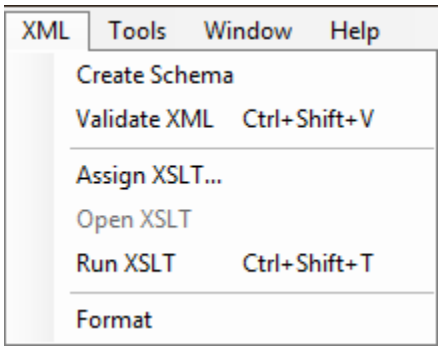
pIndentSpaces Number of spaces for each indentation. Defaults to 2

Returns String with formatted xml

This method does some basic formatting to xml strings to make them more human readable. Note that this is ONLY designed to work on non-formatted xml streams. If there are spaces between between the end and begin tags the formatting may be thrown off.

See the example below. Note that the indentation is done with spaces and the resulting XML is only meant for human reading, not to be passed to xml parsers/readers as the additional spaces will bloat the string/file considerably!

As a side note, you can load XML files into Clarion and validate them. When you load a XML file into Clarion, a new menu appears that has some options for XML files.



Example:

```
  PROGRAM
  INCLUDE('ITUtilityClass.inc'),ONCE
```

```
  MAP
  END
S   CSTRING(2000)
ITS ITStringClass
  CODE
  S = '<?xml version="1.0" encoding="utf-8"?><soap:Envelope ' &|
      'xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" ' &|
      'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ' &|
      'xmlns:xsd="http://www.w3.org/2001/XMLSchema">' &|
      '<soap:Body>' &|
      '<TermOfPayment_GetAllResponse xmlns="http://thedomain.com">' &|
      '<TermOfPayment_GetAllResult>' &|
      '<TermOfPaymentHandle><Id>1</Id></TermOfPaymentHandle>' &|
      '<TermOfPaymentHandle><Id>2</Id></TermOfPaymentHandle>' &|
      '<TermOfPaymentHandle><Id>4</Id></TermOfPaymentHandle>' &|
      '<TermOfPaymentHandle><Id>5</Id></TermOfPaymentHandle>' &|
      '</TermOfPayment_GetAllResult>' &|
      '</TermOfPayment_GetAllResponse>' &|
      '</soap:Body></soap:Envelope>'

  S = ITS.FormatXML(S,2)
  ITS.WriteStringToFile('test.xml',S)
```

This would result in test.xml file that looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd
="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <TermOfPayment_GetAllResponse xmlns="http://thedomain.com">
      <TermOfPayment_GetAllResult>
        <TermOfPaymentHandle>
          <Id>1</Id>
        </TermOfPaymentHandle>
        <TermOfPaymentHandle>
          <Id>2</Id>
        </TermOfPaymentHandle>
        <TermOfPaymentHandle>
          <Id>4</Id>
        </TermOfPaymentHandle>
        <TermOfPaymentHandle>
          <Id>5</Id>
        </TermOfPaymentHandle>
      </TermOfPayment_GetAllResult>
    </TermOfPayment_GetAllResponse>
  </soap:Body>
</soap:Envelope>
```

See also:
WriteStringToFile 316

3.32.5.17 FreeCSVFields — String Class - Methods

Prototype: **PROTECTED**

This method is protected and should not be called outside the class. It disposes of data in the CSVFields 284, Frees the queue and then disposes of it.

See also:
ParseCSVLine

3.32.5.18 FreeLines

This method frees the Lines 284 queue and de-allocates the memory for each line.

Example:
```
ITS  ITStringClass
 Code
 ITS.AddLine('First line',True)
 ITS.AddLine('Second')
 ITS.AddLine('Third')
 ITS.FreeString(True)
 !! Lines still has 3 records in it
 ITS.FreeLines
 !! Lines is now freed
```

See also:
Lines 284

AddLine 288

3.32.5.19 FreeString

Prototype: **(Byte pWords=0)**

pWords Indicates if the Words queue should be freed also

This method frees the Lines and/or the Words 286 queue. If the pWords is set to true it only frees the Words 286 queue. If it is false, then both the Words queue and the Lines 284 queue are freed.

Example:
```
ITS  ITStringClass
 Code
 ITS.AddLine('First line',True)
 ITS.AddLine('Second')
 ITS.AddLine('Third')
 ITS.FreeString(True)
 !! Lines still has 3 records in it
 ITS.FreeString
 !! Lines is now freed
```

See also:
Lines 284

AddLine 288

3.32.5.20 GetFieldPrefix

Prototype: **(String pFieldName),String**

pFieldName Name of field/column to split

Returns Prefix of the field

This method uses the SplitFieldName 310 method to split the prefix from the field name. It returns the prefix without the trailing colon. This can be very useful when parsing Clarion code or TXA/TXD code to extract field information.

Example:

```
ITS  ITStringClass
Fn   String(20)
 Code
 Fn = 'MYF:SysID'
 Message('Prefix of ' & Fn & ' = ' & ITS.GetFieldPrefix(Fn))
```

See also:
CombineFieldName 290
SplitFieldName 310

3.32.5.21 GetLine

Prototype: **(Long pIndex),String**

pIndex Queue pointer for the record to get. This must be in the range of 1 to Records(Lines)

Returns The contents of the Lines.OL for the record being pointed to or an empty string if the pIndex value is out of range.

This method simply returns the contents of a specified line in the Lines queue.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 ITS.AddLine('First line',True)
 ITS.AddLine('Second line')
 ITS.AddLine('Third line')
 S = ITS.GetLine(1)   ! S = 'First line'
 S = ITS.GetLine(2)   ! S = 'Second line'
 S = ITS.GetLine(3)   ! S = 'Third line'
```

See also:
Lines 284
AddLine 288

3.32.5.22 GetStringBetween

Prototype:	**(STRING pBegin, STRING pEnd, STRING pSearchString,<Long pBeginPos>, <Long pEndBeginPos>, BYTE pCaseSensitive=FALSE),STRING**
pBegin	The string to search for to mark the beginning of the new string. The new string starts in the character position **after** the pBegin is found.
pEnd	The string to search for to mark the end of the new string. The new string ends in the character position **before** the pEnd is found.
pSearchString	The string to search in.
pBeginPos	Optionally specify where to start searching in pSearchString to find the beginning position
pEndBeginPos	Optionally specify where to start searching in pSearchString to find the beginning of the end position.
pCaseSensitive	Optionally specify if the search should be case sensitive. Defaults to false (i.e. NOT case sensitive - all comparisons are done using UPPER()
Returns	Returns the string found between pBegin and pEnd or empty string if either or neither was found or if the end position is lower than the beginning position.

This method takes a string and then searches for a beginning and end strings in it and returns the string between those two strings. This is very handy for example to extract data from for example html or xml strings.

Example:

```
GetServiceXMLError        PROCEDURE(STRING pXML)
ITS          ITStringClass
Err_Value    CSTRING(101)
Err_Text     CSTRING(1025)
  CODE
  !! (STRING pXML),STRING
  IF INSTRING('<SOAP:REASON>',UPPER(pXML),1,1)
    Err_Value = ITS.GetStringBetween('<soap:Value>',
'</soap:Value>',pXML)
    Err_Text  = ITS.GetStringBetween('<soap:Text xml:lang="en">',
'</soap:Text>', pXML)
    ITS.ODS('Value:  ' & Err_Value)
    ITS.ODS('Reason: ' & Err_Text)
  ELSIF INSTRING('<FAULTCODE>',UPPER(pXML),1,1)
    Err_Value = ITS.GetStringBetween('<faultcode>',   '</faultcode>',
pXML)
    Err_Text  = ITS.GetStringBetween('<faultstring>', '</faultstring>',
pXML)
    ITS.ODS('Fault Code:  ' & Err_Value)
    ITS.ODS('Fault String: ' & Err_Text)
  END
  RETURN(Err_Text)
```

See also:

InsertString 301

RemoveHTML 308

3.32.5.23 GetWord

Prototype: **(Long pIndex),String**

pIndex Queue pointer for the record to get. This must be in the range of 1 to Records(Words)

Returns The contents of the Words.Word for the record being pointed to or an empty string if the pIndex value is out of range.

This method simply returns the contents of a specified word in the Words queue.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = 'This is a fun exercise'
 ITS.StringToWords(S,,,False) ! Do not sort alphabetically
 S = ITS.GetWord(1)
 !! S would now be 'This'
 S = ITS.GetWord(2)
 !! S would now be 'is'

 S = 'This is a fun exercise'
 ITS.StringToWords(S) ! Sort alphabetically
 S = ITS.GetWord(1)
 !! S would now be 'a'
 S = ITS.GetWord(2)
 !! S would now be 'fun'
 Message(S)
```

See also:
Lines 284
AddLine 288

3.32.5.24 Insert String

Prototype: **(STRING pStringToAdd, STRING pStringToAddTo, LONG pPosition),STRING**

pStringToAdd The string to add into pStringToAddTo. It can be of any length.

pStringToAddTo The string to add pStringToAdd into. It can be of any length.

pPosition Position in pStringToAddTo where pStringToAdd should be inserted. This can be any value from 1 to LEN(pStringToAddTo)

Returns The contents of the insertion of pStringToAdd into pStringToAddTo.

This method allows you to insert one string into another at a given position.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = 'Text insert to'
 S = ITS.InsertString( ' to',S, 4)
 S = ITS.InsertString( 'in', S,15)
 !! "Text to insert into"
```

See also:
Lines 284

AddLine 288

3.32.5.25 LinesToFile

Prototype:	**(String pFileName, <String pEOL>),LONG,PROC**
pFileName	File to write the lines to.
pEOL	Optional End Of Line delimiter to use. Defaults to CR+LF <13,10> if either pEOL is omitted or is empty.
Returns	Optionally returns the number of lines in the Lines 284 queue property.

This method simply calls the WriteQToFile 316. It is provided as perhaps a better name for the method.

Example:

```
ITS  ITStringClass
S    String(1024)
  Code
  ITS.AddLine('Icetips Alta LLC')
  ITS.AddLine('3430 E Highway 101, Ste. 28')
  ITS.AddLine('Port Angeles, WA 98362')
  ITS.LinesToFile('C:\Temp\Test.txt')
```

This will write the records in the queue tot he C:\Temp\Test.txt file as:

```
Icetips Alta LLC
3430 E Highway 101, Ste. 28
Port Angeles, WA 98362
```

See also:
WriteQToFile 316

3.32.5.26 LinesToString

String Class - Methods

Prototype: **(String pEOL),LONG**

pEOL Specify an EOL character or characters. If it is not specified <13,10> CRLF will be used.

Returns Integer with the total length of the string

This method takes all the lines in the Lines property queue and concatenates them into a single string and places it in the LineString 283 property. Use StringFromLines 312 to get the string returned.

Example:

See also:
StringFromLines 312

3.32.5.27 MatchParenthesis

String Class - Methods

Prototype: **(String pS),Short**

pS String to test

Returns Number of mismatching parenthesis

This method counts opening parenthesis and closing parenthesis and returns the difference. If the parenthesis are matched, the return value should be zero. If there are more opening parenthesis than closing parenthesis, the return value will be a positive number. If there are more closing parenthesis than opening parenthesis, the return value will be negative.

Example:

```
ITS ITStringClass
S   String(1024)
  Code
  S = '1*(123/(12/4))+(23*54)'
  Message('MatchParenthesis = ' & ITS.MatchParenthesis(S))  ! Should return
0
  S = '1*(123/(12/4))+(23*54'
  Message('MatchParenthesis = ' & ITS.MatchParenthesis(S))  ! Should return
1
  S = '1*123/(12/4))+(23*54'
  Message('MatchParenthesis = ' & ITS.MatchParenthesis(S))  ! Should return
-1
```

See also:
AddIntoParenthesis 287

3.32.5.28 ParseDelimtedLine

Prototype:	**(STRING pLine, STRING pDelimiter, BYTE pRemoveQuotes=TRUE,<STRING pQuote>)!!,LONG,PROC**
pLine	The line to parse
pDelimiter	Delimeter to search for
pRemoveQuotes	If TRUE it removes the pQuote character from the first and last position in the word(s)
pQuote	Quote character to look for and remove. Defaults to double quote (")
Returns	Returns number of "words" parsed out of the pLine string.

This method takes a string with a single line of delimited data and parses it into the Words 286 queue. Please note the wrong spelling of delimited in the method name. In January 2016, we added a correctly spelled method, ParseDelimitedLine, that simply calls this method.

Example:

Given a file called Test.txt:

```
"Name","Title","FirstName","MI","LastName","Empty","Address","City","State"
"Mr. Terry Gilley","Mr.","Terry","","Gilley","","1425 N Field St   Apt
1111","Austin","TX"
"Mr. John Davis","Mr.","John","","Davis","","1825 Adolph St","El Paso","TX"
```

```
Program
 Map
 End
 Include('ITUtilityClass.inc'),ONCE


FN  CString(1025)
ITS  ITStringClass
I    Long
 Code
 FN = 'Test.txt'
 ITS.FileToString(FN)
 ITS.ODS('File len: ' & Len(ITS.FileString))
 ITS.StringToLines(ITS.FileString)
 ITS.ODS('Lines: ' & Records(ITS.Lines))
 Loop I = 1 To Records(ITS.Lines)
   Get(ITS.Lines,I)
   ITS.ODS('Line ' & Format(I,@n3) & ': ' & ITS.Lines.OL)
   ITS.ParseDelimtedLine (ITS.Lines.OL, ',',,'"')
 End
```

Would yield 3 lines and each line would be parsed into 9 words or records in the ITS.Words queue.

See also:

ParseCSVLine 305

FileToString 295

StringToLines 313

3.32.5.29 ParseCSVLine

Prototype:	**(STRING pLine, Byte pStringsAreQuoted=TRUE,<String pDelimiter>, <STRING pQuoteChar>),LONG**
pLine	The line to parse. It should contain valid CSV data, either comma separated or semicolon separated.
pStringsAreQuoted	Indicates if the strings are quoted with double quotes. Defaults to TRUE.
pDelimiter	Optionally set the delimiter to use (single character). This is the character that separate the fields in the CSV line. If it is not specified, comma (,) is used.
pQuoteChar	Optionally set the character used to quote strings. If not specified double quotes (") is used. Added 2015-09-12
Returns	Number of fields parsed out of the line.

This method takes a string and parses out comma separated (CSV) field data from it. Normally this data is formatted with commas separating the fields, strings inside double quotes and numbers unquoted. The method takes into account if the delimiter is inside quoted strings and will ignore them. This will only work properly in well formatted CSV data string.

Example:

Parsing a normal comma separated file:

```
ITS  ITStringClass
S    STRING(255)
I    LONG
F    LONG
X    LONG
  CODE
  ITS.FileToLines('testcsv.csv')
  LOOP I = 1 TO Records(ITS.Lines)
    GET(ITS.Lines,I)
    F = ITS.ParseCSVLine(ITS.Lines.OL)
    ITS.ODS('Line: ' & ITS.Lines.OL)
    IF F
      ITS.ODS('  ' & Format(F,@n_3) & ' Fields from ' & ITS.Lines.OL)
      LOOP X = 1 TO F
        GET(ITS.CSVFields,X)
        ITS.ODS('    ' & Format(X,@n02) & ' ' & ITS.CSVFields.OL)
      END
    END
  END
```

Parsing a **semicolon separated** file:

```
ITS  ITStringClass
S    STRING(255)
I    LONG
F    LONG
X    LONG
  CODE
  ITS.FileToLines('testcsv2.csv')
  LOOP I = 1 TO Records(ITS.Lines)
```

```
    GET(ITS.Lines,I)
    F = ITS.ParseCSVLine(ITS.Lines.OL,False,';')
    ITS.ODS('Line: ' & ITS.Lines.OL)
    IF F
      ITS.ODS('  ' & Format(F,@n_3) & ' Fields from ' & ITS.Lines.OL)
      LOOP X = 1 TO F
        GET(ITS.CSVFields,X)
        ITS.ODS('    ' & Format(X,@n02) & ' ' & ITS.CSVFields.OL)
      END
    END
  END
```

See also:
FileToLines 294
CSVFields 284
Lines 284

3.32.5.30 PadString

String Class - Methods

Prototype: **(String pStr, String pPad, Short pLen, Byte pStart=0),String**

pStr	String to pad
pPad	Padding character or string to use
pLen	Length of padded string
pStart	Optional where to start the padding

Returns Returns the padded string

This method can be used to pad a string with character or repeated characters. This can be useful when lining up strings with for example results of calculations when writing information to text files.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = 'Test one'
 S = ITS.PadString(S, '.',50) & '1'
 ! S should now be 'Test one.....................................1'
```

3.32.5.31 ReadFileToQ

String Class - Methods

Prototype: **(String pFileName),Long**

pFileName	Name of (text) file to read into lines

Returns Integer value containing the number of lines read into the Lines 284 property.

This method calls the FileToLines 294 method, it's just a different naming convention matching the WriteQToFile 316.

Example:

```
Fn  CString(2049)
I   Long
L   Long
ITST ITStringClass
 Code
 Fn = 'c:\temp\textfile.txt
 If Exists(ShortPath(Fn))
   L = ITST.ReadFileToQ(Fn)
   Loop I = 1 To L
     ITST.ODS(Format(I,@n4) & ' ' & ITST.GetLine(I))
   End
 End
```

See also:
FileToLines 294
WriteQToFile 316
StringToLines 313

3.32.5.32 ReadFileToString

String Class - Methods

Prototype: **(String pFileName),LONG,PROC**

pFileName Filename to read

Returns Number of bytes read.

This is a very powerful method that uses API to read a file into a dynamic string containing the entire file. The file is read into a single buffer so the reading is extremely fast. While this is primarily designed for text files, this method could be used to read just about any kind of file. Once the file is read the contents is stored in the FileString 284 property where you can manipulate it. Because of this you need to make sure that you do not have important data in the FileString property before you call this method.

Example:

```
ITS  ITStringClass
 Code
 ITS.ReadFileToString('c:\temp\test.txt')
 !! You can now access the contents of test.txt by using the ITS.FileString
property.
```

See also:
FileString 284
WriteStringToFile 316
WriteQToFile 316
AllocateFileString 289
DisposeFileString 293

3.32.5.33 RemoveHTML

Prototype: **(String pHTMLString),String**

pHTMLString String to remove HTML tags from

Returns String that has been cleared of HTML tags

This method will remove all HTML tags from the string that is passed in. This is very convenient if you need to scrape just the text off of a web page that you have loaded from file or from the web. Everything that is inside < ... > tags is removed along with the "<" and ">" characters.

Example:

```
ITS  ITStringClass
S    String(4000)
 Code
 S = '<html><body>This is a test web page</body></html'
 S = ITS.RemoveHTML(S)
 Message('Stripped HTML = ' & S)
```

S now contains "This is a test web page"

See also:
HTMLString 284

3.32.5.34 SetDepunctuationString

Prototype: **(String pDepunct)**

pDepunct String of punctuation characters to replace with spaces when using the DepunctuateString method

This method is used to set the DepunctuationString 283 property which is used in the DepunctuateString 292 method to determine what characters to replace with space. This is very useful when using the StringToWords 314 to control what characters are considered to split words. Note that this method **replaces** the punctuation string used to determine punctuation.

Example:

```
ITS  ITStringClass
 Code
 ITS.SetDepunctionationString('/;:|')
 ITS.DepunctuateString('This;that:other|there)
 !! By setting the depunctuation string the ;: and | in the string will be
replaced with a space.
```

3.32.5.35 SetLineValue

Prototype: **(String pLine)**

pLine The new value to use for the currently selected line

This method is used to set the text for the currently selected line, for example after a call to the GetLine 299 method. This will replace the data that was in the selected line

Example:

```
ITS  ITStringClass
 Code
 ITS.AddLine('One',True)
 ITS.AddLine('Two')
 ITS.AddLine('Three')
 If ITS.GetLine(2)
   ITS.SetLineValue('Twenty')
 End
```

The text in the second line is now "Twenty" instead of "Two".

See also:
AddLine 288
AppendToLine 289
GetLine 299
Lines 284

3.32.5.36 SetStringBetween

String Class - Methods

Prototype:	**(STRING pBegin, STRING pEnd, STRING pSearchString, STRING pInsertString, BYTE pHandling, <Long pBeginPos>, <Long pEndBeginPos>, BYTE pCaseSensitive=FALSE),STRING**
pBegin	The string to search for to mark the beginning of the new string. The new string starts in the character position **after** the pBegin is found.
pEnd	The string to search for to mark the end of the new string. The new string ends in the character position **before** the pEnd is found.
pSearchString	The string to search in.
pInsertString	The string to insert into pSearchString
pHandling	How the insert is handled. It can be ITStIns:Prefix 283, ITStIns:Append 283 or ITStIns:Replace 283.
pBeginPos	Optionally specify where to start searching in pSearchString to find the beginning position
pEndBeginPos	Optionally specify where to start searching in pSearchString to find the beginning of the end position.
pCaseSensitive	Optionally specify if the search should be case sensitive. Defaults to false (i.e. NOT case sensitive - all comparisons are done using UPPER()
Returns	Returns the updated string with the pInsertString inserted if the pBegin and pEnd were found.

This method inserts a substring into a string given a start string and end string. Note that this method does NOT do multiple inserts, it will only insert the string in one place, the first place it finds between the pBegin and pEnd strings. The SetStringEnd 284 and SetStringFound 285 can be checked to see if there are more occurrences to replace and where they begin. If SetStringFound 285 is 0 then there are no more pBegin/pEnd pairs in the resulting string.

Example:

```
 PROGRAM
  INCLUDE('ITUtilityClass.inc'),ONCE
  MAP
  END
S    CString(1000)
I    Long
ITS  ITStringClass
  CODE
  S = '<a>stuff</a>'
  S = ITS.SetStringBetween('<a>','</a>',S,'test',ITStIns:Append)
  ! String is now: "<a>stufftest</a>"


  S = '<a>stuff</a>'
  S = ITS.SetStringBetween('<a>','</a>',S,'test',ITStIns:Prefix)
  ! String is now: "<a>teststuff</a>"

  S = '<a>stuff</a>'
  S = ITS.SetStringBetween('<a>','</a>',S,'test',ITStIns:Replace)
  ! String is now: "<a>test</a>"
```

See also:
GetStringBetween 300

3.32.5.37 SetSplitStringProgress

String Class - Methods

Prototype: **(Long pProgressFEQ)**

pProgressFEQ A FEQ (Field EQuate label) of a progress bar to use when splitting strings using the SplitString 311 method.

This method sets the SplitStringProgressFEQ 285 property with a FEQ of a progress control, which is used in the SplitString 311 method to indicate progress through the string it is splitting. Normally SplitString is very fast so this is not needed at all, but if you are dealing with large strings then it is nice for the user to see a progress bar. Note that the splitting of the string is in a tight loop to make it as fast as possible.

See also:
SplitString 311
SplitStringProgressFEQ 285

3.32.5.38 SplitFieldName

String Class - Methods

Prototype: **(String pFieldName, <*String pPrefix>),String**

pFieldName The label of the field/column to split
pPrefix Optional string parameter to receive the prefix of the field/column label

Returns The field/column name without prefix

This method takes a field/column label and splits it up into the field/column name and the prefix. If the pPrefix parameter is not omitted it is assigned to the prefix of the field/column label.

Example:

```
ITS  ITStringClass
Fn   String(20)
Pf   String(5)
 Code
 Fn = 'MYF:SysID'
 Fn = ITS.SplitFieldName(Fn,Pf)
```

At this point Fn will contain 'MYF' and Fn will contain 'SysID'

See also:

CombineFieldName 290

GetFieldPrefix 299

3.32.5.39 SplitString

String Class - Methods

Prototype:	**(String pS, String pDelimiter, BYTE pDelimiterStartsLine=FALSE),Long,PROC**
pS	String to split
pDelimiter	Delimiter to use to split the string
pDelimiterStartsLine	Indicates that the delimiter is the start of the line, not end of it. Defaults to FALSE. 2015-06-23
Returns	Number of lines created from the split string

This method is primarily used to split a string containing linebreaks into a queue of lines. But it can also be used to separate text based on other delimiters.

Example:

```
ITS  ITStringClass
S    String(1024)
I    Long
 Code
 S = 'Icetips Creative, Inc.<13,10>136 E. 8th Street<13,10>Port Angeles, WA
98362'
 ITS.SplitString(S,'<13,10>)
 Loop I = 1 To Records(ITS.Lines)
   Message('Line ' & I & ITS.GetLine(I))
 End

 ! Example showing where a delimiter starts the line
 S ='SVM2,"1","1"' &|
     'SVM2,"1","2"' &|
     'SVM2,"1","2"'
  ITS.SplitString(S,'SVM2',TRUE)
  ITS.ODS('Lines: ' & Records(ITS.Lines))
  Loop I = 1 To Records(ITS.Lines)
    Get(ITS.Lines,I)
```

```
    ITS.ODS('Line ' & Format(I,@n02) & ' ' & ITS.Lines.OL)
  End
```

This would show 3 records in the ITS.Lines queue.

See also:
Lines 284
StringToLines 313
StringToWords 314

3.32.5.40 StringFromLines

String Class - Methods

Prototype: **(),STRING**

Returns String from SELF.Lines

This method constructs a string from the Lines queue and returns it.

Example:

```
ITS  ITStringClass
SQL  CString(1025)
  CODE
  ITS.AddLine('SELECT TOP(1) CAST(MonthStartDateTime AS DATE) AS
MonthStartDate',TRUE)
  ITS.AddLine('  FROM dbo.PropertyToDo')
  ITS.AddLine(' ORDER BY MonthStartDateTime DESC')
  SQL = ITS.StringFromLines()
```

The resulting string will look like this:

```
SELECT TOP(1) CAST(MonthStartDateTime AS DATE) AS MonthStartDate
  FROM dbo.PropertyToDo
 ORDER BY MonthStartDateTime DESC
```

This makes it easy to construct SQL statements for example

See also:
LinesToString 303
LineString 283

3.32.5.41 StringToFile

String Class - Methods

Prototype: **(String pFileName, String pContent, Byte pAppend=False)!!,Long,PROC**

pFileName Name of the file to write to

pContent Contents to write to the file. This can be any data, binary or text.

pAppend Indicates if the method should append the data to the file if it exists. Defaults to False

Returns Optionally returns the number of bytes written to the file.

This method is a wrapper for the WriteStringToFile. It simply provides an alternate and shorter name.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = 'Icetips Alta LLC<13,10>' &|
     '3430 E Highway 101 Ste 28<13,10>' &|
     'Port Angeles, WA 98362<13,10>'
 ITS.StringToFile('C:\Temp\Test.txt')
```

This will write the contents of the string to a file called C:\Temp\Test.txt and it will now contain this:

```
Icetips Alta LLC
3430 E Highway 101 Ste 28
Port Angeles, WA 98362
```

See also:

3.32.5.42 StringToLines — String Class - Methods

Prototype: **(String pS, <String pDel>),LONG,PROC**

pS String to parse into lines

pDel Optional delimiter to use as End-Of-Line string. Added 2015-09-12

Returns Number of records in the Lines property

This method calls the SplitString method in order to split the string using CRLF (<13,10>) as delimiter.

Example:

```
ITS  ITStringClass
S    String(1024)
I    Long
 Code
 S = 'Icetips Creative, Inc.<13,10>136 E. 8th Street<13,10>Port Angeles, WA
98362'
 ITS.StringToLines(S)
 Loop I = 1 To Records(ITS.Lines)
   Message('Line ' & I & ': ' & ITS.GetLine(I))
 End
```

See also:

Lines 284

FileToLines 294

SplitString 311

StringToWords 314

3.32.5.43 StringToWords

String Class - Methods

Prototype:	**(String pS, Byte pCount=True, Byte pCaseSensitive=False, Byte pSort=True),Long,PROC**
pS	String to parse into words
pCount	Reserved
pCaseSensitive	Indicates if the case of the string should be preserved. Set to false by default, meaning that all words are lower case
pSort	If true the words are sorted alphabetically. If false the words are not sorted and are added in order so the words in the queue are in the same order as they are in the string. Defaults to True.
pAllowDigits	If true then words made up of digits are counted as words. If false words made up of digits are not counted. Defaults to False. (Added: January 19, 2013)
Returns	Returns the number of words in the string

This method is used to parse a string into words. First all punctuation is removed by using the DepunctuateString 292 method. Note that ONLY unique words are stored along with a word counter, so this is not suitable for parsing words in order to put the text back together. This is very useful to get words out of a document and count each occurance of it in the document.

NOTE: On September 18, 2009 the 4th parameter was added to make it possible to extract a non-sorted list of the words in the string.

On January 19, 2013 the 5th parameter was added to allow digit words to be counted.

Example:

```
ITS  ITStringClass
S    String(1024)
I    Long
 Code
 S = 'This and that is this 123'
 ITS.StringToWords(S) !! Returns 5, excludes the '123'
 ITS.StringToWords(S,,,,True)  !! Returns 6, includes the '123'
 Loop I = 1 To Records(ITS.Words)
   Message('Word ' & I & ': ' & ITS.GetWord(I))
 End
```

The first call would result in 5 words being added to the Words queue, "this", "and", "that" and "is" and the word "this" would have a count of 2 while the others would have a count of 1. The second call would result in 6 words being added, including the numeric word at the end, see pAllowDigits.

See also:

Words 286

ITWordQ 282

DepunctuateString 292

SetDepunctuationString 308
DepunctuationString 283

3.32.5.44 StripParenthesis

String Class - Methods

Prototype: **(String pTxt, <String pParLeft>,<String pParRight>),String**

pTxt String to parse parenthesis out of
pParLeft Optional character for left parenthesis (opening)
pParRight Optional character for right parenthesis (closing)

Returns Returns contents of pTxt without opening/closing parenthesis

This method removes an opening and closing pair of parenthesis from the beginning and end of string. The left and right parenthesis can be set to different characters if the contents needs to be removed from square brackets for example.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = '(123+x)'
 Message('Stripped: ' & ITS.StripParenthesis(S))
 ! Would show '123+x'
 S = '[APPLICATION]'
 Message('Stripped: ' & ITS.StripParenthesis(S))
 ! Would show 'APPLICATION'
```

See also:
MatchParenthesis 303
AddIntoParenthesis 287

3.32.5.45 UseEither

String Class - Methods

Prototype: **(String pS1, String pS2, Byte pFavourite=1),String**

pS1 First string
pS2 Second string
pFavourite Favourite string

Returns Selected string

This method chooses between two strings based on which should be a favourite. If pS1 is not empty and it is the favourite, it is returned, otherwise pS2 is returned if it is not empty. If pS1 is empty and pS2 is the favourite, it will return pS2. Otherwise the function returns an empty string. This method was developed for a special situation and only really makes sense to use with variables rather than literal strings.

Example:
```
ITS  ITStringClass
S    String(100)
 Code
 S = ITS.UseEither(CON:AddressMain,CON:AddressShip,2)
```

This would assign the CON:Addr1Addl to the S variable, since it favours the second parameter. If the second parameter is empty, it would use the first one. In this case it determines if a shipping address should be used if it's specified.

3.32.5.46 WriteQToFile

Prototype: **(String pFileName, <String pDel>)**

pFileName	Name of the file to write the queue of Lines to.
pDel	Optional delimiter to use as End-Of-Line string. Added 2015-09-12

This method writes the lines in the Lines queue to the specified file by allocating memory for all the entries in the queue, then creating a dynamic string which is then written to the file using the WriteStringToFile 316 method.

Example:
```
ITS  ITStringClass
S    String(1024)
  Code
  ITS.AddLine('Icetips Alta LLC')
  ITS.AddLine('3430 E Highway 101, Ste. 28')
  ITS.AddLine('Port Angeles, WA 98362')
  ITS.WriteQToFile('C:\Temp\Test.txt')
```

This will write the records in the queue tot he C:\Temp\Test.txt file as:

```
Icetips Alta LLC
3430 E Highway 101, Ste. 28
Port Angeles, WA 98362
```

See also:
Lines 284
WriteStringToFile 316

3.32.5.47 WriteStringToFile

Prototype: **(String pFileName, String pContent, Byte pAppend=False),Long,PROC**

pFileName	Name of file to write the string to
pContent	Content to write to the file
pAppend	If pAppend is true, the file is opened or created if it doesn't exist. If it is false, the file is always created.

Returns Returns the number of bytes written to the file

This method writes the passed string to the specified file. The content can be anything, but would normally be written to a text file, although there is nothing preventing this from being used with binary data files. You can specify if the file should be overwritten or if it should be appended to. With pAppend=False, the file will always be created, even if it exists. With pAppend=True, the file will be written to but never overwritten. If the file doesn't exist it will be created when pAppend=True. This method uses the AllocateFileString to allocate memory for the contents and by doing so re-allocates the FileString. Because of that you need to make sure that you do not have important data in the FileString property before you call this method.

Example:

```
ITS  ITStringClass
S    String(1024)
 Code
 S = 'Icetips Alta LLC<13,10>' &|
     '3430 E Highway 101 Ste 28<13,10>' &|
     'Port Angeles, WA 98362<13,10>'
 ITS.WriteStringToFile('C:\Temp\Test.txt')
```

This will write the contents of the string to a file called C:\Temp\Test.txt and it will now contain this:

```
Icetips Alta LLC
3430 E Highway 101 Ste 28
Port Angeles, WA 98362
```

See also:
FileString 284
ReadFileToString 307
WriteQToFile 316
AllocateFileString 289
DisposeFileString 293

3.32.5.48 Construct — String Class - Methods

The String class constructor creates instances of the Lines and Words queues. It also constructs the DepunctuationString 283 property which by default contains ()[]{{}!@#$%^&*-

See also:
Lines 284
Words 286
Destruct 317
DepunctuationString 283
SetDepunctuationString 308

3.32.5.49 Destruct — String Class - Methods

The String class destructor de-allocates memory for the Lines 284 and Words 286 queues as well as for TempS 286 and ResStr 286 dynamic strings. It also calls the DisposeFileString method which disposes of the FileString property.

See also:
Lines 284
Words 286
TempS 286
ResStr 286
FileString 284
Construct 317
DisposeFileString 293

3.32.6 Tutorial String Class

String Class *(ITStringClass)*

The String Class contains 35 methods as of version 1.1.2397 that was released on May 4, 2011. The String class has some very powerful methods to work with strings, such as read a whole file into a string or write a string to a file. This can be used to read and write large files very, very fast.

Below are short tutorials on how to accomplish various things with this class. The tutorials are short and demonstrate simple use of the methods.

Reading and writing text files

The String class has a method, ReadFileToString 307, to read a text file directly into a string buffer that is dynamically allocated by the string and is big enough to read the entire file into it. No file drivers or clarion files are used, rather the method uses API to open the file and read from it. Reading the file is extremely fast and once read, the entire content of the file is stored in the FileString 284 property of the String class.

Example:
```
ITS  ITStringClass
  Code
  ITS.ReadFileToString('C:\Temp\Filename.txt')
```

Once this code executes, the entire contents of C:\Temp\Filename.txt is in the ITS.FileString property and you can do with it what you want.
A related method is the WriteStringToFile. This is the opposite of ReadFileToString as it takes a string and writes it to a file.

Example:
```
ITS  ITStringClass
S    String(1024) ! 1K buffer
  Code
  S = 'This is a test<13,10>writing to a text file!'
  ITS.WriteStringToFile('C:\Temp\Filename.txt',S)
```

This will create a text file that would look like this:

This is a test
writing to a text file!

Export comma separated files

There is a third method to write to files, WriteQToFile 316, that is used along with the AddLine 288 method to build up a text file. This is very convenient for example to create all kinds of export files, i.e. comma separated files to transfer data from one program to another.

Example:

```
ITS  ITStringClass
Fn   CString(2049)
  Code
  Set(CUS:NameKey)
  ITS.AddLine('Name,Email,Address,State,PostalCode,Country',True)
  Loop
    If Access:Customers.Next() <> LEVEL:Benign
      Break
    End
    ITS.AddLine('"' & Clip(CUS:Name) & '",' &|
                '"' & Clip(CUS:Email) & '",' &|
                '"' & Clip(CUS:Address) & '",' &|
                '"' & Clip(CUS:State) & '",' &|
                '"' & Clip(CUS:PostalCode)  & '",' &|
                '"' & Clip(CUS:Country) & '"')
  End
  If FileDialog('Save CSV file',Fn,'Export files (*.csv)|*.csv|All files
(*.*)|*.*',|
                FILE:Save + FILE:KeepDir + FILE:LongNames)
    ITS.WriteQToFile(Fn)
  End
```

As with the ReadFileToString 307 and WriteStringToFile 316 there are no file drivers involved, just API codes that makes this very fast. The first line is added right before the loop. The second parameter tells the class to free the queue before it starts adding to it, i.e. by using True as the second parameter you are telling the class that this is the first line. Each loop through the data file adds a single line to the queue that holds the lines, ITS.Lines. After the loop is done going through the data a Save dialog is opened to get the filename for the file and the contents of the queue are written to the selected file. Note that you can place CR+LF charcters into the line string thus building up more than one line to write into the text file.

Export XML files

Exporting to XML is certainly possible using the String Class. Since xml files are much more flexible than CSV files the way you do it completely depends on how you need the xml file to look.

Example:

```
ITS  ITStringClass
Fn   CString(2049)
I    Long
  Code

  ITS.AddLine('',True)  !! First line

  Loop I = 1 To 10
    ITS.AddLine('  <item>' & I & '</item>')
  End
  If FileDialog('Save XML',Fn,'XML (*.xml)|*.xml',FILE:Save + FILE:KeepDir +
FILE:LongName)
    ITS.WriteQToFile(Fn)
  End
```

This produces a very simple xml that will look like this:

```
<?xml version="1.0" ?>
  <item>1</item>
  <item>2</item>
  <item>3</item>
```

```
  <item>4</item>
  <item>5</item>
  <item>6</item>
  <item>7</item>
  <item>8</item>
  <item>9</item>
  <item>10</item>
```

Obviously this is an extremely simple example, but it demonstrates how you could construct xml one line at the time, without too much effort.

Create lines from strings

One of the features of the string class is the ability to split a string into lines using the SplitString 311 method. This means that you can make it split a text file that you have read with ReadFileToString 307 into lines and then use the lines rather than use the string buffer directly.

Example:

```
ITS  ITStringClass
I    Long
Q    Queue
OneLine CString(1025)
     End

Window WINDOW('Lines'),AT(,,260,100),GRAY
       LIST,AT(13,10,234,81),USE(?List1),VSCROLL,|
         FONT('Courier New',9,,),FORMAT('1024L(2)'),FROM(Q)
     END

 Code
 ITS.ReadFileToString('Export.clw')
 ITS.SplitString(ITS.FileString,'<13,10>')
 ITS.DumpLinesInQ(Q,Q.OneLine)
 Open(Window)
 Display
 Accept
 End
 Close(Window)
 Free(Q)
```

This code reads in a file called Export.clw and displays it in a listbox. Note that you need version 1.1.2352 or later of the Icetips Utilities for this code to work as the DumpLinesInQ 293 was added in 1.1.2352. This code uses the SplitString 311 to split the string using the CR+LF as delimiter, i.e. to indicate where the string should split. Note that you cannot use the ITS.Lines queue directly in a listbox because the data is contained in a reference variable. Hence the need for a local queue and the DumpLinesInQ 293 method.

You can use the SplitString 311 to split any string using any delimiter that you want, for example the pipe character, '|', if you have a pipe delimited string that you want to split up.

Example:

```
ITS  ITStringClass
I    Long
S    String(100)
X    CString(101)
R    Long
 Code
 S = 'First|Second|Third|Fourth|Fifth'
 ITS.SplitString(S,'|')
 R = Records(ITS.Lines)
 Loop I = 1 To R
   Get(ITS.Lines,I)
   X = X & ITS.Lines.OL & Choose(I<R,' + ','')
```

```
End
Message('New string = ' & X)
```

Removing spaces and punctuation from strings

The String class has a method called CompactString 291. It can be very useful for all kinds of purposes, particularly as a sort field in queues where names or words may be typed differently. It is also invaluable when comparing strings that may be typed differently when the difference is not just the case, which can be worked around with Upper() or Lower(). All punctuation and spaces are removed from the string and it can optionally be set to all upper case to eliminate case sensitivity.

Example:

```
ITS  ITStringClass
S1    String(100)
S2    String(100)

 Code
 S1 = 'This is a test to show how a string is compacted!'
 S2 = 'This.is.a.test.to.show.how.a.string.is.compacted!'
 If S1 = S2
   Message('String S1 and S2 are the same before Compacting')
 Else
   Message('String S1 and S2 are NOT the same before Compacting|' & S1 & '|' & S2)
 End
 S1 = ITS.CompactString(S1,True)
 S2 = ITS.CompactString(S2,True)
 If S1 = S2
   Message('String S1 and S2 are the same after Compacting|' & S1 & '|' & S2)
 Else
   Message('String S1 and S2 are NOT the same after Compacting|' & S1 & '|' & S2)
 End
```

Here I used a period between each word to make them different. The text is exactly the same. Those two strings are not equal in comparison and obviously will show the "String S1 and S2 are NOT the same before Compacting" message. After the compacting however, since all the punctuation and spaces are gone the two strings match as: "THISISATESTTOSHOWHOWASTRINGISCOMPACTED"

3.33 Thread Limit Class

3.33.1 Overview — Thread Limit Class

This class is used on a procedure that should be limited to a fixed number of instances that can be active at the same time. See the Thread Limit Global Class 330 for more information. See also the Icetips Utilities Global extension template 451 and the Icetips Thread Limit procedure extension template 471.

```
ITThreadLimitClass       CLASS(ITUtilityClass),TYPE,Module('ITThreadLimitClass.
clw'),Link('ITThreadLimitClass.clw',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
Win 323                          &Window
GlobalClass 322                  &ITThreadLimitGlobalClass 330
ProcedureName 323                CString(256)
MaxRuns 323                      Byte
InstanceToSelect 322             Byte !! Ranging between 1 to MaxRuns
TheThread 323                    Long
RestoreWindowOnActivation 323    Byte

ActivateWindow 323               Procedure !! Called from global class
Init 214                         Procedure(String pProcedureName)
AddProcedure 324                 Procedure(ITThreadLimitGlobalClass 168 pGlobalClass,
WINDOW pWin),Byte
RemoveProcedure 326              Procedure
SetProcedureLimit 327            Procedure(Long pMaxRun)
SetInstanceToSelect 327          Procedure(Long pInstanceToSelect)
TakeLimit 328                    Procedure(),VIRTUAL
CloseWindowHandler 325           Procedure(),BYTE
Construct 82                     Procedure
                         END
```

3.33.2 Properties — Thread Limit Class

The Thread Limit Class has 8 properties to keep track of various things. This includes the instance of the Thread Limit Global Class 330 which is used by this class.

```
Win 323                          &Window
GlobalClass 322                  &ITThreadLimitGlobalClass 330
ProcedureName 323                CString(256)
MaxRuns 323                      Byte
InstanceToSelect 322             Byte !! Ranging between 1 to MaxRuns
TheThread 323                    Long
RestoreWindowOnActivation 323    Byte
```

3.33.2.1 GlobalClass — Thread Limit Class - Properties

This is a reference to an instance of the Thread Limit Global Class 330 that this class uses. It is passed in with a call to the AddProcedure 324 method.

3.33.2.2 InstanceToSelect — Thread Limit Class - Properties

Stores the instance to activate when the MaxRuns 323 limit is reached. This property is not used yet and the instance defaults to the last instance run.

3.33.2.3 MaxRuns — Thread Limit Class - Properties

This property is used to set the maximum number that a procedure can be run without hitting a limit. It is set with the SetProcedureLimit 327 method.

3.33.2.4 ProcedureName — Thread Limit Class - Properties

Name of the procedure.

3.33.2.5 RestoreWindowOnActivation — Thread Limit Class - Properties

Boolean flag that indicates is the window should be restored when it is activated if the window is minimized. It defaults to TRUE so iconized window will be restored when the window is activated in the ActivateWindow 323 method.

3.33.2.6 TheThread — Thread Limit Class - Properties

The thread number of the current thread.

3.33.2.7 Win — Thread Limit Class - Properties

Reference to the window of the procedure.

3.33.3 Methods — Thread Limit Class

Enter topic text here.

ActivateWindow 323	Procedure !! Called from global class
Init 214	Procedure(String pProcedureName)
AddProcedure 324	Procedure(ITThreadLimitGlobalClass 168 pGlobalClass, WINDOW pWin),Byte
RemoveProcedure 326	Procedure
SetProcedureLimit 327	Procedure(Long pMaxRun)
SetInstanceToSelect 327	Procedure(Long pInstanceToSelect)
TakeLimit 328	Procedure(),VIRTUAL
CloseWindowHandler 325	Procedure(),BYTE
Construct 82	Procedure

3.33.3.1 ActivateWindow — Thread Limit Class - Methods

Prototype: **(none)**

This method is called from the ActivateThread 332 method of the Thread Limit Global Class 330. It sets the window as foreground window, activates it and if it is minimized it restores it to its original size.

See also:
ActivateThread 332

3.33.3.2 AddProcedure

Prototype: **(WINDOW pWin),BYTE**

pWin Reference to the procedure window

Returns If the procedure was added successfully the method returns True. If the procedure hit the MaxRuns 323 limit, the method returns False.

This method attempts to add the procedure to the class. If it fails then the procedure limit has been reached - this will normally never happen as that is taken care of by the CheckProcedure 324 method. This method also registers the CloseWindowHandler 325 method to clean up when the window closes. The template generate the call to this method into the WindowManager.Init method at priority 8005 so it is executed just after the window is opened as it needs the handle to the window to save it in case it needs to be activated when the procedure hits the MaxRuns 323 limit.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(1)
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End
```

See also:

AddProcedure 332

CheckProcedure 324

RemoveProcedure 333 (Global class)

RemoveProcedure 326 (This class)

3.33.3.3 CheckProcedure

Prototype: **(ITThreadLimitGlobalClass pGlobalClass),BYTE**

pGlobalClass Reference to the global class instantiated in the project or application.

Returns If the procedure can be run without hitting the MaxRuns 323 limit, the method returns True. If the procedure hits the MaxRuns 323 limit, the method returns

False.

This method is used to check at the very start of the procedure, right after INIT 326 and SetProcedureLimit 327, if the procedure is going to hit its MaxRuns limit or not. If it hits the limit the method returns False but if it does hit the limit it returns True. The template generate the call to this method into the WindowManager.Run method at priority 1 so it is the very first code that is executed when the procedure is run.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(1)
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End
```

See also:

AddProcedure 324

Init 326

SetProcedureLimit 327

CheckProcedure 333 (Global class)

3.33.3.4 CloseWindowHandler

Prototype: **(),BYTE**

Returns Returns LEVEL:Benign

This method is a handler for the EVENT:CloseWindow and removes the procedure and the window from the Thread Limit Class.

See also:

Init 326

3.33.3.5 Init

Prototype: **(String pProcedureName)**

pProcedure Name of the procedure to limit.

This method must be called before calling the AddProcedure 324 method. It also sets the ProcedureName 323 property. The template generate the call to this method into the WindowManager.Run method at priority 1 so it is the very first code that is executed when the procedure is run.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(1)
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End
```

See also:
CloseWindowHandler 325
AddProcedure 324
ProcedureName 323

3.33.3.6 RemoveProcedure

Prototype: **(none)**

This method removes the window from the Thread Limit Global Class 330. It is called by the CloseWindowHandler 325 as part of its clean up process.

See also:
CloseWindowHandler 325
AddProcedure 324

3.33.3.7 SetInstanceToSelect

Prototype: **(Long pInstanceToSelect)**

pInstanceToSelect Instance number to select when window is activated if multiple instances are allowed

This method sets the InstanceToSelect 322 property. Note that this is not currently implemented and the last instance is the default instance to be selected.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(5)
 ITThreadLocal.SetInstanceToSelect(5)  !! Last instance
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End
```

See also:
InstanceToSelect 322

ActivateWindow 323

3.33.3.8 SetProcedureLimit

Prototype: **(Long pMaxRun)**

pMaxRun Maximum number of instances that can be active at the same time.

This method is used to set the MaxRuns 323 property, which controls how many instances of the procedure can be active at the same time. The template generate the call to this method into the WindowManager.Run method at priority 1 so it is the very first code that is executed when the procedure is run.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
```

```
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(5)
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End
```

In this case up to 5 different threads with the "BrowsePeople" procedure could be active at the same time. When the 6th instance is started the AddProcedure 324 will return False indicating that the MaxRun 323 limit has been reached.

See also:
MaxRun 323

AddProcedure 324

3.33.3.9 TakeLimit

Thread Limit Class - Methods

Prototype: **(none),VIRTUAL**

This method is designed to be overridden in the project or application. The Icetips Thread Limiter template 471 generate 3 embed points, one in the data section and two in the code section, one before and one after the generated PARENT.TakeLimit call. The PARENT method calls the ActivateThread 332 method of the global class which in turns calls the ActivateWindow 323 method of the local class.

Example:

```
ITThreadLocal  Class(ITThreadLimitClass)
TakeLimit        Procedure,VIRTUAL
               End

ThisWindow.Run  PROCEDURE
 Code
 ITThreadLocal.Init('Browsepeople')
 ITThreadLocal.SetProcedureLimit(5)
 If Not ITThreadLocal.CheckProcedure(ITThreadLimit)
   Return LEVEL:Fatal
 End
 ReturnValue = PARENT.Run()
 RETURN ReturnValue


ThisWindow.Init PROCEDURE
 Code
 If Not ITThreadLocal.AddProcedure(QuickWindow)
   Return LEVEL:Fatal
 End

ITThreadLocal.TakeLimit        Procedure!,VIRTUAL
 Code
 SetCursor()  !! Clear cursor if the "Icetips Hide Windows while loading" template
is active.
 Message('You can only run ' & SELF.MaxRuns & ' instances of the ' &
```

```
SELF.ProcedureName)
 PARENT.TakeLimit
```

This shows how you can place a message before PARENT.TakeLimit to take the appropriate action.

See also:
ActivateThread 332
ActivateWindow 323

3.33.3.10 Construct — Thread Limit Class - Methods

Prototype: **(none)**

The constructor sets the value of InstanceToSelect 322 to 1, MaxRuns 323 to 1 and RestoreWinodwOnActivation 372 to True:

```
SELF.InstanceToSelect          = 1
SELF.MaxRuns                   = 1
SELF.RestoreWindowOnActivation = True
```

See also:
Init

3.34 Thread Limit Global Class

3.34.1 Overview — Thread Limit Global Class

The Thread Limit Global class inherits from the Utility Class. It is used, along witht he Thread Limit Class, to keep track of procedures that should be limited to 1 or more maximum instances. When that limit is reached no more instances of the particular procedure can be started and the last instance of the procedure is brought into focus. See the Thread Limit Class 322 for information about the class that is used on the procedure. The global class is declared globally and is called by the procedure class. See also the Icetips Utilities Global extension template 451 and the Icetips Thread Limit procedure extension template 471.

```
ITThreadLimitGlobalClass CLASS(ITUtilityClass),TYPE,Module('ITThreadLimitClass.clw'),Link('ITTh
CriticalSection 331             &ICriticalSection 330
WindowThreads 331               &ITGlobalTLQ 330
WindowHandles 331               &ITInstanceQ 331

ActivateThread 332              Procedure(String pProcedureName, Long pInstanceToSelect)
AddProcedure 324                Procedure(WINDOW pWin, String pProcedureName, ITThreadLimitClass 3
RemoveProcedure 326             Procedure(String pProcedureName)
RemoveWindowHandle 333          Procedure(LONG pHandle)
SetProcedureLimit 327           Procedure(String pProcedureName, Long pMaxRun)
Construct 334                   Procedure
Destruct 334                    Procedure
                            END
```

3.34.2 Data Types — Thread Limit Global Class

The Thread Limit Global Class uses 3 data types.

ICriticalSection 330
ITGlobalTLQ 330
ITInstanceQ 331

3.34.2.1 ICriticalSection — Thread Limit Global Class - Data Types

The ICriticalSection is an interface declared in CWSYNCH.INT:

```
ICriticalSection    INTERFACE(ISyncObject)
Wait                  PROCEDURE(),DERIVED
Release               PROCEDURE(),DERIVED
Kill                  PROCEDURE(),DERIVED
                    END
```

Please refer to the Clarion documentation and help for more information. This is used to protect non threaded global data that is used by the Thread Limit Global class.

3.34.2.2 ITGlobalTLQ — Thread Limit Global Class - Data Types

This is a Typed Queue that extends the ITGlobalThreadQ:

```
ITGlobalThreadQ         QUEUE,TYPE
ThreadNumber              Long
ProcedureName             CString(256)
WindowHandle              Long
```

```
WindowClientHandle        Long
InsertLevel               Long
IsFrameWindow             Byte
WindowReference           &Window
                        END

ITGlobalTLQ             QUEUE(ITGlobalThreadQ),TYPE,PRE(ITGlobalTLQ)
ProcRunCnt                Long  !! Procedure Run Counter.  Counts how many
instances of a procedure are running
MaxNumber                 Long  !! Maximum number of instances running.
ProcedureNameUpperCase    CString(256)
ProcClass                 &ITThreadLimitClass
                        END
```

The ITGlobalTLQ is used to create an instance of the queue in the class to keep track of threading information.

3.34.2.3 ITInstanceQ — Thread Limit Global Class - Data Types

This is a typed queue that keeps a pointer to the queue derived from ITGlobalTLQ 330.

```
ITInstanceQ             QUEUE,TYPE
ProcedurePtr              Long       !! Pointer from ITGlobalTLQ
WindowHandle              IT_HWND    !! Handle of the window
                        END
```

3.34.3 Properties — Thread Limit Global Class

The Thread Limit Global Class has 3 properties:

CriticalSection 331	&ICriticalSection 330
WindowThreads 331	&ITGlobalTLQ 330
WindowHandles 331	&ITInstanceQ 331

3.34.3.1 CriticalSection — Thread Limit Global Class - Properties

This is an instance of the ICriticalSection 330 interface that is used to make the access to the global queues thread safe.

3.34.3.2 WindowHandles — Thread Limit Global Class - Properties

This is an instance of the ITInstanceQ 331 that keeps track of each window handle for each instance of a thread limited procedure that has been opened.

3.34.3.3 WindowThreads — Thread Limit Global Class - Properties

This is an instance of the ITGlobalTLQ 330 that keeps track of procedures, threads etc. for each limited procedure.

3.34.4 Methods — Thread Limit Global Class

The Thread Limit Global Class has 7 methods, including constructor and destructor. None of those methods should be called by code, only methods from the Thread Limit Class 322 implementation which calls the appropriate methods in the Thread Limit Global Class. For this reason there is no example code for any of those methods. Note that the Procedure name is used as a unique identifier for the procedure. It is stored in the WindowThreads queue, but for searching and comparison it is

stored as upper case, so GETting from the queue is case insensitive.

ActivateThread 332	Procedure(String pProcedureName, Long pInstanceToSelect)
AddProcedure 324	Procedure(WINDOW pWin, String pProcedureName, ITThreadLimitClass 3
RemoveProcedure 326	Procedure(String pProcedureName)
RemoveWindowHandle 333	Procedure(LONG pHandle)
SetProcedureLimit 327	Procedure(String pProcedureName, Long pMaxRun)
Construct 334	Procedure
Destruct 334	Procedure

3.34.4.1 ActivateThread

Prototype: **(String pProcedureName, Long pInstanceToSelect)**

pProcedureName Name of the procedure.

pInstanceToSelect Number of the instance to select if the procedure count has reach the limit.

This method is called by the TakeLimit 328 method in the Thread Limit Class 322. It in turn calls the ActivateWindow 323 method in the Thread Limit Class 322 to activate the correct window.

See also:
TakeLimit 328
ActivateWindow 323

3.34.4.2 AddProcedure

Prototype: **(WINDOW pWin, String pProcedureName, ITThreadLimitClass pProcClass, Byte pMaxRuns),BYTE**

pWin Reference to the procedure window.

pProcedureName Name of the procedure.

pProcClass Instance of the Thread Limit Class 322.

pMaxRuns Maximum number of runs for this procedure.

Returns If the procedure was added successfully the method returns True. If the procedure hit the MaxRuns 323 limit, the method returns False.

This method is called by the AddProcedure 324 method of the Thread Limit Class 322. It is used to add or update the procedure listing in the global class. If the procedure name already exists, the ProcRunCnt field in the WindowThreads 331 queue is incremented.

This method is called after the window is opened and the RemoveProcedure 333 method is called when it is closing.

See also:
AddProcedure 324
CheckProcedure 333

RemoveProcedure 333
MaxRuns 323

3.34.4.3 CheckProcedure

Prototype: **(String pProcedureName, Byte pMaxRuns),BYTE**

pProcedureName Name of the procedure.

pMaxRuns Maximum number of runs for this procedure.

Returns If the procedure can be run without hitting the MaxRuns 323 limit, the method returns True. If the procedure hits the MaxRuns 323 limit, the method returns False.

This method is called at the start of the procedure to check if this procedure is reaching the MaxRuns limit. If it is equal to the MaxRuns 323 property as indicated by the pMaxRuns parameter the method returns false indicating that the limit has been reached.

See also:
AddProcedure 332
RemoveProcedure 333
CheckProcedure 324

3.34.4.4 RemoveProcedure

Prototype: **(String pProcedureName)**

pProcedureName Name of the procedure to remove

This method is called when the window closes and removes the procedure from the WindowThreads 331 queue if there is only one instance of the procedure. If there are multiple instances, the ProcRunCnt is decremented. This method is called by the RemoveProcedure 326 method in the Thread Limit Class 322.

See also:
AddProcedure 332 (Thread Limit Global Class)
AddProcedure 324 (Thread Limit Class)
RemoveProcedure 326

3.34.4.5 RemoveWindowHandle

Prototype: **(LONG pHandle)**

pHandle Handle (hWnd) for the window to remove.

This method is used to remove a window handle from the WindowHandles 331 queue. It keeps track of the windows that have been opened with the same procedure on different threads so they can be activated when the MaxRuns 323 limit has been reached. This method is called by the RemoveProcedure 333 method.

See also:
RemoveProcedure 333
AddProcedure 332 (Thread Limit Global Class)
AddProcedure 324 (Thread Limit Class)

3.34.4.6 SetProcedureLimit
Thread Limit Global Class - Methods

Prototype: **(String pProcedureName, Long pMaxRun)**

pProcedureName Name of the procedure to set the limit for

pMaxRuns Number to limit the runs to

This method is called by the SetProcedureLimit 327 method of the Thread Limit Class 322. It sets the maximum number of runs that the procedure can reach before hitting the limit.

See also:
SetProcedureLimit 327
MaxRuns 323

3.34.4.7 Construct
Thread Limit Global Class - Methods

Prototype: **None**

The constructor instantiates the 3 properties, CriticalSection 331, WindowHandles 331 and WindowThreads 331.

See also:
Destruct 334

3.34.4.8 Destruct
Thread Limit Global Class - Methods

Prototype: **None**

This method destroys and disposes of the WindowHandles 331 and WindowThreads 331 queues.

See also:
Construct 334

3.35 Utility Class

3.35.1 Overview Utility Class

The Utility Class derives from the Windows Class 365. It adds several lower level functions.

The Utility class has several very useful functions to do various things, such as create nested directories, format error message strings with or without file errors and many more. The Utility Class has it's own Construct 354 and Destruct 412 methods that set up queues that are used by the class.

```
ITUtilityClass
Class(ITWindowsClass),TYPE,Module('ITUtilityClass.clw'),Link('ITUtilityClass',_ITUt
ilLinkMode_),DLL(_ITUtilDllMode_)


MSQ 338                          &IT_MS_Q
MultiFileSelPath 338             CString(1025)
FontName 337                     CString(65)
FontSize 338                     Long
FontColor 337                    Long
FontStyle 338                    Long
FontCharset 337                  Long


CheckOplocks                Procedure(),Byte
ColorToHtml 340                  Procedure(Long pColorValue),String
CompareCRC32 341                 Procedure(String pBuffer, Ulong pCRC),Byte
CreateDirectories 342            Procedure(String pDirectories, String
pStartDir),Long,PROC ! Returns number of directories created
DirectoryExists 342              Procedure(String pDirectory),Byte  ! Returns
true/false if the directory exists.
ErrorMsg 343                     Procedure(Byte pStdErr=True, Byte pFilErr=False,
<String pSeparator>),String
FirstNonSpace 344                Procedure(String pS),Long
GetCRC32 346                     Procedure(String pBuffer),Ulong
GetClockFromString 344           Procedure(String pClock),Long  ! Takes clock value
formatted in 'hh:mm:ss' and returns 1/100 seconds.
GetClockValue 345                Procedure(Long pClock, Byte pIntervalMin, Byte
pRoundUp),Long
GetCommandLineParam 345          Procedure(String pFlag),String
GetExcelDate 346                 Procedure(Long pClarionDate),Long
GetFileInfo 347                  Procedure(String pFileName, Long pAtt=0, <*ANY pDate>,
<*ANY pTime>, <*Long pSize>, <*Long pAttrib>)
GetFormatted100sec 348           Procedure(Long pTime,<String pDelimiter>,<String
pTimeFormat>),String
GetHour 348                      Procedure(Long pClock),Long
GetMinute 349                    Procedure(Long pClock),Long
GetUnixDateTime 349              Procedure(*DECIMAL pUnixTime, <*Long pTime>),Long
HtmlToColor 350                  Procedure(String pHtmlColor),Long
LongToHex 73                     Procedure(Long pLong),String
MultiFileSelect 350              Procedure(String pMfS),Long,PROC
SelectFont 351                   Procedure(String pCaption,<*String pFontName>,<*Long
pFontSize>,<*Long pFontColor>,<*Long pFontStyle>,<*Long pFontCharset>),Byte
SetOplocksOff               Procedure(),Byte

Construct 412                    Procedure
Destruct 412                     Procedure
                          End
```

3.35.2 Equates — Utility Class

The Utility class uses the following Equates in splitting file and path information:

```
FNS_Drive                EQUATE(01h)
FNS_Path                 EQUATE(02h)
FNS_File                 EQUATE(04h)
FNS_Ext                  EQUATE(08h)
FNS_FullPath             EQUATE(FNS_Drive+FNS_Path)
FNS_FileName             EQUATE(FNS_File+FNS_Ext)
```

When using the GetFilePart 65 method these equates are used to determine what parts of the filename are returned.

Example

```
ITU  ITUtilityClass

FN   CString(1025)
FP   CString(1025)
 Code
 FN = 'C:\Clarion6\LibSrc\ABFILE.CLW'
 FP = ITU.GetFilePart(FNS_Drive)          ! Returns 'C:'
 FP = ITU.GetFilePart(FNS_Path)           ! Returns '\Clarion6\LibSrc\'
 FP = ITU.GetFilePart(FNS_File)           ! Returns 'ABFILE'
 FP = ITU.GetFilePart(FNS_Ext)            ! Returns '.CLW'

 FP = ITU.GetFilePart(FNS_File+FNS_Ext)   ! Returns 'ABFILE.CLW'
 FP = ITU.GetFilePart(FNS_Drive+FNS_Path) ! Returns 'C:\Clarion6\LibSrc\'
```

3.35.3 Data Types — Utility Class

The Utility class uses one special data type for multi file selections.

IT_MS_Q 336

See also:
MultiFileSelect 350

3.35.3.1 IT_MS_Q — Utility Class - Data Types

The following queue type is used in Multi File Selection.

```
IT_MS_Q                  QUEUE,TYPE
MSFileN                    CString(1025)
                         END
```

See also:
MultiFileSelect 350

3.35.4 Properties — Utility Class

The Utility class has seven public properties.

MSQ 338	&IT_MS_Q
MultiFileSelPath 338	CString(1025)
FontName 337	CString(65)
FontSize 338	Long
FontColor 337	Long
FontStyle 338	Long
FontCharset 337	Long

See also:
MultiFileSelect 350
SelectFont 351

3.35.4.1 FontCharset — Utility Class - Properties

This is a Long used in the SelectFont methods to hold the name of the selected character set for the font.

FontCharset Long

See also:
SelectFont 351

3.35.4.2 FontColor — Utility Class - Properties

This is a Long used in the SelectFont methods to hold the Clarion color value for the font.

FontColor Long

See also:
SelectFont 351

3.35.4.3 FontName — Utility Class - Properties

This is a CString used in the SelectFont methods to hold the name of the selected font.

FontName CString(65)

See also:
SelectFont 351

3.35.4.4 FontSize — Utility Class - Properties

This is a Long used in the SelectFont methods to hold the point size value for the font.

```
FontSize                    Long
```

See also:
SelectFont 351

3.35.4.5 FontStyle — Utility Class - Properties

This is a Long used in the SelectFont methods to hold the font style value for the font. The style could be for example FONT:Bold or FONT:Italic. See the FONT topic in the Clarion help for more information about the FONT: equates.

```
FontStyle                   Long
```

See also:
SelectFont 351

3.35.4.6 MSQ — Utility Class - Properties

Queue used in Multi file Select. It is declared as:

```
MSQ                         &IT_MS_Q 336
```

See also:
MultiFileSelect 350

3.35.4.7 MultiFileSelPath — Utility Class - Properties

This is a CString used in the Multi File Select methods to hold the path for the files.

```
MultiFileSelPath            CString(1025)
```

See also:
MultiFileSelect 350

3.35.5 Methods — Utility Class

The Utility class has 20 methods, including the Constructor and Destructor.

```
ColorToHtml 340             Procedure(Long pColorValue),String
CompareCRC32 341            Procedure(String pBuffer, Ulong pCRC),Byte
CreateDirectories 342       Procedure(String pDirectories, String
```

```
pStartDir),Long,PROC ! Returns number of directories created
DirectoryExists 342          Procedure(String pDirectory),Byte  ! Returns
true/false if the directory exists.
ErrorMsg 343                 Procedure(Byte pStdErr=True, Byte pFilErr=False,
<String pSeparator>),String
FirstNonSpace 344            Procedure(String pS),Long
GetCRC32 346                 Procedure(String pBuffer),Ulong
GetClockFromString 344       Procedure(String pClock),Long  ! Takes clock value
formatted in 'hh:mm:ss' and returns 1/100 seconds.
GetClockValue 345            Procedure(Long pClock, Byte pIntervalMin, Byte
pRoundUp),Long
GetCommandLineParam 345      Procedure(String pFlag),String
GetExcelDate 346             Procedure(Long pClarionDate),Long
GetFileInfo 347              Procedure(String pFileName, Long pAtt=0, <*ANY pDate>,
<*ANY pTime>, <*Long pSize>, <*Long pAttrib>)
GetFormatted100sec 348       Procedure(Long pTime,<String pDelimiter>,<String
pTimeFormat>),String
GetHour 348                  Procedure(Long pClock),Long
GetMinute 349                Procedure(Long pClock),Long
GetUnixDateTime 349          Procedure(*DECIMAL pUnixTime, <*Long pTime>),Long
HtmlToColor 350              Procedure(String pHtmlColor),Long
LongToHex 73                 Procedure(Long pLong),String
MultiFileSelect 350          Procedure(String pMfS),Long,PROC
SelectFont 351               Procedure(String pCaption,<*String pFontName>,<*Long
pFontSize>,<*Long pFontColor>,<*Long pFontStyle>,<*Long pFontCharset>),Byte

Construct 412                Procedure
Destruct 412                 Procedure
```

3.35.5.1 CheckOplocks

Utility Class - Methods

Prototype: **(),Byte**

Returns Returns true if all the registry keys are correct. Returns false if any of them is wrong.

This method checks 4 registry keys for the opportunistic locking (OpLock) settings. Those keys are:

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/MRXSmb/Parameters/OplocksDisabled = 1 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/EnableOpLocks = 0 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/EnableOpLockForceClose = 1 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/CachedOPenLimit = 0 (hex - DWORD)

When OpLock is turned on it can cause data corruption and lost data. The SetOplocksOff 353 method can be used to turn the opportunistic locking off, but it must be called elevated so I would suggest adding a small executable file that does that and you can call it to fix the opportunistic locking, as described in the example below.

Example:

```
ITU  ITUtilityClass
ITS  ITShellClass
I    Byte
 Code
 Loop
   If Not ITU.CheckOplocks
     Message('The Oportunistic Locking is not set correctly and can cause data
corruption.','OPLOCK ERROR',ICON:Hand)
     ITS.ITRun('FixOplocks.exe',True,,,True)
   End
   I+=1
   If I > 4
     Message('Cannot fix Oportunistic Locking.  Aborting','OPLOCK ERROR',ICON:Hand)
     HALT(0)
   End
 End
```

See also:
SetOplocksOff 353

http://support.microsoft.com/kb/296264

http://www.icetips.com/showarticle.php?articleid=264

3.35.5.2 ColorToHTML — Utility Class - Methods

Prototype: **(Long pColorValue),String**

pColorValue Clarion color value

Returns String containing the correct html color value as #RRGGBB

This method takes an ordinary Clarion 24bit color value and turns it into a standard HTML color string in the format '#RRGGBB' See the TestUtilityClass procedure in the Example program 514

Example:

```
ITU        ITUtilityClass
Col        Long
HTMLColor  String(7)
 Code
 If ColorDialog('Select color',Col)
   HTMLColor = ITU.ColorToHTML(Col)
   ?HTMLColor {Prop:FontColor} = Col
   Display
 End
```

See also:
HTMLToColor 350

3.35.5.3 ColorToRGB — Utility Class - Methods

Prototype: **(Long pColor, <*Long pRed>,<*Long pGreen>, <*Long pBlue>)**

pColor	Color to split into RGB
pRed	Optional parameter to return the **Red color**
pGreen	Optional parameter to return the **Green color**
pBlue	Optional parameter to return the **Blue color**

This method takes a Clarion color and splits it into Red, Green and Blue values.

Example:

```
ITU  ITUtilityClass
R   LONG
G   LONG
B   LONG
Col LONG
  CODE
  Col = COLOR:Red
  ITU.ColorToRGB(Col, R, G, B)
  IF ITU.RGBToColor(R, G, B) <> Col
    MESSAGE('Something is wrong, color is not returned correctly')
  ELSE
    MESSAGE('Got the correct color back!')
  END
```

See also:

ColorToHTML 340

RGBToColor 351

RGBToHSL

3.35.5.4 CompareCRC32 — Utility Class - Methods

Prototype: **(String pBuffer, Ulong pCRC),Byte**

pBuffer String to compare CRC value for

pCRC The CRC value to compare the CRC from the buffer to.

Returns True or false depending on if the CRC values match.

This methods takes a string buffer, calculates the CRC value for it and then compares it to the passed CRC value. If the calculated and passed CRC match the method returns True. If they do not match, the method returns False.

Example:

```
ITU            ITUtilityClass
Loc:CRC        ULong
Loc:TestString String(20)
Loc:CRCString  String(20)
 Code
 Loc:TestString = 'Icetips Creative'
 Loc:CRCString  = 'Icetips Creative'

 Loc:CRC = ITU.GetCRC32(Loc:TestString)
 If ITU.CompareCRC32(Loc:CRCString,Loc:CRC)
```

```
   Message('The strings match:)' &|
           '|TestString = ' & Loc:CRC & |
           '|CRCString  = ' & ITU.GetCRC32(Loc:CRCString),'Strings
match:)',ICON:Exclamation)
 Else
   Message('The strings do NOT match:(' &|
           '|TestString = ' & Loc:CRC  &|
           '|CRCString  = ' & ITU.GetCRC32(Loc:CRCString),'Strings do NOT match:(',
ICON:Hand)
 End
```

See also:
GetCRC32 346

3.35.5.5 CreateDirectories

Prototype: **(String pDirectories, String pStartDir),Long**

pDirectories String containing the path to create below the start directory. Any directory or directories that do not exist will be created by the method. The string does not need to start with a backslash.

pStartDir String containing the full path to the start directory. This directory must exist.

Returns The number of directories created.

This is a very powerful function that will create as many nested directories as you want.

Example:

```
CurrentUser   CString(101)
ITU           ITUtilityClass
 Code
 ITU.CreateDirectories('Data\Temp',Path())
 CurrentUser = 'John'
 ITU.CreateDirectories('User\' & CurrentUser & '\Data\Temp\Stuff',Path())
```

In the first example, if Path() is C:\Clarion this would result in C:\Clarion\Data\Temp to be created. In the second example, if Path() is C:\Clarion this would result in C:\Clarion\John\Data\Temp\Stuff to be created.

This method is extremely useful when multiple levels of directories is needed. If any of the directories in pDirectories does not exist, it will be created. Directories in pStartDir will not be created, i.e. pStartDir must exist.

Note: *In Beta 2, there was potentially dangerous code in this method that changed the path using SetPath() This code has been removed in the Beta 3 release and this method should be perfectly safe.*

3.35.5.6 DirectoryExists

Prototype: **(String pDirectory),Byte**

pDirectory Full path to the directory to check.

Returns True or false depending on if the directory exists or not

This method uses the Exists function to check if the directory exists or not.

Example:

```
P    CString(1025)
ITU  ITUtilityClass
 Code
 P = 'C:\Clarion\Apps\MyApp'
 If ITU.DirectoryExists(P)
   Copy ('myfile.txt',P & 'myfile.txt')
 End
```

See also:
IsFolder 71

3.35.5.7 ErrorMsg

Utility Class - Methods

Prototype: **(Byte pStdErr=True, Byte pFilErr=False, <String pSeparator>),String**

pStdErr Indicates if error information from ErrorCode() and Error() should be included in the error string. This defaults to True

pFilErr Indicates if error information from FileErrorCode() and FileError() should be included in the string. This defaults to False.

[pSeparator] Indicates a separator string used between the standard errors and the file errors. If omitted it defaults to a single space character.

Returns String containing formatted error message.

This method returns a formatted error message in the format of:

(ErrorCode) Error Separator (FileErrorCode) FileError

If you are going to show the error string in a Message() function, then you could pass '<13,10>' as separator to put the standard error and file error on separate lines.

Note:
When working with SQL drivers any SQL errors that come from the backend database engine trigger errorcode 90. In that case the FileErrorCode() and FileError() will contain extended error information that comes from the database engine. If you are dealing with SQL engines, always pass True as the second parameter to make sure that you will get that extended error information. Otherwise you will simply get error code 90 - File Driver Error, which doesn't really tell you anything about the actual problem. If you know that you are only going to be getting errors from the database engine, i.e. error 90, then you can simply leave the first parameter, pStdErr as false. Then this method will only return the extended error information from the backend.

Example:

```
ITU  ITUtilityClass
 Code
 Add(MyFile)
 Case ErrorCode()
 Of 90
```

```
  Message('SQL Error: ' & ITU.ErrorMsg(False,True)
Else
  Message('Error: ' & ITU.ErrorMsg(True,True,'<13,10>')
End
```

3.35.5.8 FirstNonSpace

Prototype: **(String pS),Long**

pS String to check

Returns The first non space character position in the string pS

This method uses the internal StrSpn function to find the first non-space character in the string passed as pS. For more information about StrSpn please refer to the web, for example http://www.cplusplus.com/ref/cstring/strspn.html

Example:

```
S    String(255)
I    Long
ITU  ITUtilityClass
 Code
 S = ' This is a string'
 I = ITU.FirstNonSpace(S)
```

In this case I would be equal to 2.

3.35.5.9 GetClockFromString

Prototype: **(String pClock),Long**

pClock String formatted as HH:MM:SS

Returns Time value in 1/100 seconds.

This function takes a string formatted as HH:MM:SS and returns the time value from it, i.e. hundredths of seconds elapsed from midnight. See the Clock() function in Clarion.

Example:

```
S    String(10)
C    Long
ITU  ITUtilityClass
 Code
 S = '13:54:10'
 C = ITU.GetClockFromString(S)
```

C would now be (13*60*60*100) + (54*60*100) + (10*100) or 5,005,000

See also:
GetClockValue 345

3.35.5.10 GetClockValue

Utility Class - Methods

Prototype: **(Long pClock, Byte pIntervalMin, Byte pRoundUp),Long**

pClock Time value.

pIntervalMin Interval in minutes.

pRoundUp Rounds up if True or down if false.

Return Time value after rounding up or down to the given interval.

This function takes a clock value and rounds it up or down to the nearest clock indicated in the pIntervalMin. This is very useful for schedule type of calculation where an appointment should be scheduled for example with 15 minute intervals starting at the hour, i.e. at 18:00, 18:15, 18:30 and 18:45. This function makes it very easy to do that. Simply give it the time, interval in minutes and if it should round up or down and it will return the correct time value.

Example:

```
C1   Long
C2   Long
I    Byte
R    Byte
ITU  ITUtilityClass
 Code
 C1 = ITU.GetClockFromString('18:16:00')
 I = 15 ! 15 minute interval
 R = True
 C1 = ITU.GetClockValue(C1,I,R)
 I = 15 ! 15 minute interval
 R = False
 C2 = ITU.GetClockValue(C1,I,R)
```

C1 would be equal to 18:30:00 but C2 would be equal to 18:15:00

See also:
GetClockFromString 344

3.35.5.11 GetCommandLineParam

Utility Class - Methods

Prototype: **(String pFlag),String**

pFlag Command line flag to test for.

Returns The command line flag or parameter

This function parses out a flag in the command line parameters passed when the program starts up. If an equal sign is used in the parameter, for example /N=Test, the function will return Test only.

Example:

```
C    String(255)
ITU  ITUtilityClass
```

```
 Code
 ! The program was started with myprog.exe /N=Test /F=myfile.tps /Q

 C = ITU.GetCommandLineParam('/N')   ! Will return 'Test'
 C = ITU.GetCommandLineParam('/F')   ! Will return 'myfile.tps'
 C = ITU.GetCommandLineParam('/Q')   ! Will return '/Q'
```

See also:
EXEName 55 - Coreclass
ProgPath 56 - Coreclass
ProgramCommandLine 56 - CoreClass

3.35.5.12 GetCRC32 — Utility Class - Methods

Prototype: **(String pBuffer),Ulong**

pBuffer String to calculate CRC value for.

Returns The CRC32 value for the buffer string.

This methods takes a string buffer and calculates and returns the CRC value for it.

Example:

```
Loc:CRC        ULong
Loc:TestString String(20)
Loc:CRCString  String(20)
ITU            ITUtilityClass
 Code
 Loc:TestString = 'Icetips Creative'
 Loc:CRCString  = 'Icetips Creative'

 Loc:CRC = ITU.GetCRC32(Loc:TestString)
 If ITU.CompareCRC32(Loc:CRCString,Loc:CRC)
   Message('The strings match:)' &|
           '|TestString = ' & Loc:CRC & |
           '|CRCString  = ' & ITU.GetCRC32(Loc:CRCString),'String
match:)',ICON:Exclamation)
 Else
   Message('The strings do NOT match:(' &|
           '|TestString = ' & Loc:CRC  &|
           '|CRCString  = ' & ITU.GetCRC32(Loc:CRCString),'String do NOT match:(',
ICON:Hand)
 End
```

See also:
CompareCRC32 341

3.35.5.13 GetExcelDate — Utility Class - Methods

Prototype: **(Long pClarionDate),Long**

pClarionDate Standard Clarion date.

Returns Returns Excel based date value.

A date in Excel is a value that is exactly 36161 days less than the Clarion date. So this method simply subtracts that value from the standard Clarion date value and the resulting value can be used as a date value in MS Excel.

Example:

```
ED    Long
ITU   ITUtilityClass
 Code
 ED = ITU.GetExcelDate(Today())
```

See also:
GetUnixDateTime 349

3.35.5.14 GetFileInfo — Utility Class - Methods

Prototype: **(String pFileName, Long pAtt=0, <*ANY pDate>, <*ANY pTime>, <*Long pSize>,<*Long pAttrib>)**

pFileName Full path name of the file to get information about.

pAtt Specifies the ff_ file attributes 519.

[pDate] The date of the file. This can be a LONG or a DATE variable. You must specify variable in the call or omit this parameter.

[pTime] The time of the file. This can be a LONG or a TIME variable. You must specify variable in the call or omit this parameter.

[pSize] The size of the file. You must specify variable in the call or omit this parameter.

[pAttrib] The attributes of the file. You must specify variable in the call or omit this parameter.

This method will retrieve information about the file passed as pFileName. Note that date, time, size and attributes are passed by address not value so it is up to you to create variables that are used when calling this method. pDate and pTime can be LONG or DATE/TIME variables so this will work with program variables as well as table columns from SQL tables etc.

this method does not return a value, rather it returns multiple values.

Example:

```
FD   Long
FAT  Long
FT   Long
FS   Long
FA   Long
F    CString(1025)
ITU  ITUtilityClass
 Code
 FAT = ff_:Normal
 F = 'C:\Clarion\Apps\MyTest\Test.app' ! Jan 1, 2006 at 13:01:40, 12345 bytes.
 ITU.GetFileInfo(F,FAT,FD,FT,FS,FA)
 ! FD is now equal to the date value for Jan 1, 2006
 ! FT is now equal to the clock value for '13:01:40'
```

```
! FS is now equal to 12345
! FA is now equal to ff_:Normal
```

3.35.5.15 GetFormatted100sec

Prototype: **(Long pTime,<String pDelimiter>,<String pTimeFormat>),String**

pTime Standard Clarion Time value, such as Clock()

pDelimiter The delimiter character(s) to use to separate the standard formatted time and the 1/100 fractions. The default is semicolon, ':'

pTimeFormat The time format to use, such as '@t4' The default is @t4.

Returns Returns a string that contains both the time and the 1/100 second fraction, such as "13:58:17:45"

This method is useful in profiling and debugging code.

Example:

```
S    Long
X    Long
ITU  ITUtilityClass
 Code
 S = Clock()
 !! Some code here to profile
 Loop 1000000 times
   X+= 1
 End
 Message('It took ' & ITU.GetFormatted100sec(Clock() - s) ' to finish the
calculation')
```

See also:

GetClockFromString 344
GetClockValue 345
GetHour 348
GetMinute 349
GetUnixDateTime 349

3.35.5.16 GetHour

Prototype: **(Long pClock),Long**

pClock The time value.

Returns The hour part of the time value in pClock.

This returns just the hour part of the time value passed in pClock. This is handy to have when you need to just know the hour part.

Example:

```
T    Long
H    Byte
ITU  ITUtilityClass
 Code
 T = ITU.GetClockFromString('11:10:10')
 H = ITU.GetHour(T)  ! returns 11
```

See also:

GetMinute 349
GetClockFromString 344

3.35.5.17 GetMinute — Utility Class - Methods

Prototype: **(Long pClock),Long**

pClock The time value.

Returns The minute part of the time value in pClock.

This returns just the minute part of the time value passed in pClock. This is handy to have when you need to just know the minute part.

Example:

```
T    Long
M    Byte
ITU  ITUtilityClass
 Code
 T = ITU.GetClockFromString('11:10:10')
 M = ITU.GetMinute(T)  ! returns 10
```

See also:

GetHour 348
GetClockFromString 344

3.35.5.18 GetUnixDateTime — Utility Class - Methods

Prototype: **(*DECIMAL pUnixTime, <*Long pTime>),Long**

pUnixTime Parameter containing time value from a Unix system.

[pTime] Optional parameter that receives the time part of the Unix Time.

Returns Returns Clarion compatible date value.

This method takes a Unix date time, which is the number of 1/100 seconds since January 1, 1970 and returns the correct Clarion date value. It can also optionally accept a second parameter which upon return from this method will contain a Clarion compatible time value.

Example:

```
(none)
```

See also:
GetExcelDate 346

3.35.5.19 HTMLToColor

Utility Class - Methods

Prototype: **(String pHtmlColor),Long**

pHtmlColor HTML color value in the format of #RRGGBB

Returns Standard Clarion color value.

This method takes a standard HTML color value string in the form of #RRGGBB and turns it into a standard Clarion color value.

Example:

```
HTMLCol String(7)
Col     Long
ITU             ITUtilityClass
 CODE
 Col = ITU.HTMLToColor(HTMLCol)
```

See also:
ColorToHTML 340

3.35.5.20 MultiFileSelect

Utility Class - Methods

Prototype: **(String pMfS),Long**

pMfS String containing multiple file selection from the Clarion FileDialog or FileDialogA functions.

Returns Number of files selected

This function is very useful when you use FileDialog or FileDialogA to allow users to open multiple files. Then the selection is returned in a pipe delimited string where the first filename contains the path and the rest only contains the filenames. Example:

'C:\Clarion\Apps\Tests\Myapp.app|otherapp.app|thirdapp.app'

This method splits it up and puts this into the MSQ 338 queue where each entry contains the full path and filename of each selected file.

Example:

```
Fn   CString(10001)
I    Long
ITU  ITUtilityClass
```

```
 Code
 If FileDialog('Select files',Fn,'All Files
(*.*)|*.*',FILE:KeepDir+FILE:Multi+FILE:LongName)
   ITU.MultiFileSelect(FN)
   Loop I = 1 To Records(ITU.MSQ)
     Get(ITU.MSQ,I)
     ! Do something with the filename
   End
 End
```

3.35.5.21 RGBtoColor

Utility Class - Methods

Prototype: **(Byte pRed, Byte pGreen, Byte pBlue),Long**

pRed	The Red value
pGreen	The Green value
pBlue	The Blue value

Returns Returns Clarion color value.

This method takes a RGB value split into Red, Green and Blue and combines it into a regular Clarion color value.

Example:

```
ITU  ITUtilityClass
R    LONG
G    LONG
B    LONG
Col  LONG
  CODE
  Col = COLOR:Red
  ITU.ColorToRGB(Col, R, G, B)
  IF ITU.RGBToColor(R, G, B) <> Col
    MESSAGE('Something is wrong, color is not returned correctly')
  ELSE
    MESSAGE('Got the correct color back!')
  END
```

See also:

3.35.5.22 SelectFont

Utility Class - Methods

Prototype: **(String pCaption,<*String pFontName>,<*Long pFontSize>,<*Long pFontColor>,<*Long pFontStyle>,<*Long pFontCharset>),Byte**

pCaption	Caption for Font selection dialog
pFontName	Optional parameter that receives the font name
pFontSize	Optional parameter that receives the font size

pFontColor	Optional parameter that receives the font color
pFontStyle	Optional parameter that receives the font style
pFontCharset	Optional parameter that receives the font character set

Returns	Returns true or false depending on if the Cancel button was pressed or not on the Font Dialog, same as the Clarion FONTDIALOG function does.

This method is a wrapper for the Clarion FONTDIALOG function. The difference is that you don't have to declare variables to use it! Instead you simply call the method and then you will have access to all the information that was selected by using the FontName, FontSize, FontColor, etc. properties of the class.

Example:

```
ITU            ITUtilityClass
 Code
 If ITU.SelectFont('Select font')
   ! Now you can use the Font properties of the Utilities Class to access the font
information for the selected font!
 End


!  Compare this to using the Clarion FontDialog:

FontName    String(64)
FontSize    Long
FontColor   Long
FontStyle   Long
FontCharset Long
 Code
 If FontDialog('Select font',FontName, FontSize, FontColor, FontStyle, FontCharset)
   ! Now you can use the font information returned.
 End
```

See also:
FontName 337
FontSize 338
FontColor 337
FontStyle 338
FontCharset 337

3.35.5.23 SetControlBckgrnd

Utility Class - Methods

Prototype:	**(LONG pFEQ, LONG pColor, BYTE pShowLabel=FALSE, <WINDOW pTarget>)**
pFEQ	Control to set the background color for
pColor	The color value to use
pShowLabel	Optionally show the label over the control
pTarget	Optionally specify target. Needed when using with Reports.

This method sets the background for the specified control. This can be very useful to identify controls for example on reports by color coding them.

Example:

See also:

3.35.5.24 SetOplocksOff

Utility Class - Methods

Prototype: **(),Byte**

Returns Return true if the opportunistic locking was successfully turned off.

This method sets the appropriate registry keys so that opportunistic locking (OpLocks) is turned off. Those keys are:

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/MRXSmb/Parameters/OplocksDisabled = 1 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/EnableOpLocks = 0 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/EnableOpLockForceClose = 1 (hex - DWORD)

HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/LanmanServer/Parameters/CachedOPenLimit = 0 (hex - DWORD)

When OpLock is turned on it can cause data corruption and lost data. The SetOplocksOff 353 method turns the opportunistic locking off, but it must be called elevated so I would suggest adding a small executable file that does that and you can call it to fix the opportunistic locking, as described in the example below.

Example:

```
ITU  ITUtilityClass
ITS  ITShellClass
I    Byte
 Code
 Loop
   If Not ITU.CheckOplocks
     Message('The Oportunistic Locking is not set correctly and can cause data
corruption.','OPLOCK ERROR',ICON:Hand)
     ITS.ITRun('FixOplocks.exe',True,,,True)
   End
   I+=1
   If I > 4
     Message('Cannot fix Oportunistic Locking.  Aborting','OPLOCK ERROR',ICON:Hand)
     HALT(0)
   End
 End
```

See also:
CheckOplocks 339

http://support.microsoft.com/kb/296264

http://www.icetips.com/showarticle.php?articleid=264

3.35.5.25 ShowControlLabel

Prototype: **(LONG pFEQ, LONG pBckGrColor=0FFFFFFH, <WINDOW pTarget>)**

pFEQ	The control to show the control name for
pBckGrColor	Background color for the label. By default it is set to white.
pTarget	Optionally set the target for the control. Needed when using this method on reports.

Returns Return information

Method information

Example:

See also:

3.35.5.26 Construct

Prototype: **None**

The constructor creates a new instance of the SELF.MSQ 338:

```
SELF.MSQ &= NEW IT_MS_Q
```

3.35.5.27 Destruct

Prototype: **None**

The Destructor disposes of the SELF.MSQ 338 queue:

```
If Not SELF.MSQ &= NULL
  Free(SELF.MSQ)
  Dispose(SELF.MSQ)
End
```

3.36 Version Class

3.36.1 Overview Version Class

The version class is used to retrieve version information from binary files, i.e. exe and dll files. If no language information is included in the version information, it defaults to the Windows 1252 Character Set. See http://en.wikipedia.org/wiki/Windows-1252 for more information. If multiple language information is included the Version Class will **only** retrieve the first language.

```
ITVersionClass          Class(ITShellClass),TYPE,Module('ITVersionClass.clw'),Link
('ITVersionClass',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
FileExists                Byte
FileHasVersionInfo        Byte
HideDebugViewVC           Byte
VersionInfo               &ITVersionInfoQueue
VersionNames              &ITVersionNameQueue

AddClarionResources       Procedure
AddVersionName            Procedure(String pVerName, String pUseName)
GetDisplayName            Procedure(String pKeyName),String
GetLanguageString         Procedure(Long pPtr),String
GetVersionInfo            Procedure(String pValueName),String ! Returns ValueInfo
LoadVersionNames          Procedure
PTD                       Procedure(String pS, Byte pHideDebug=False),VIRTUAL
QueryValue                Procedure(Long pPtrBuffer, *Long pBufferSize, String
pName),String ! Returns value of the valuename
RetrieveFromFile          Procedure(String pFile),Byte,PROC
RetrieveFromSelf          Procedure(),Byte

Construct                 Procedure
Destruct                  Procedure
                        End
```

3.36.2 Data Types Version Class

The Version class has two datatypes, ITVersionInfoQueue 356 and ITVersionNameQueue 356 that are used to create queues to store version information in.

```
ITVersionInfoQueue      QUEUE,TYPE
ValueName                 CSTRING(50)
ValueUseName              CSTRING(50)
ValueInfo                 CSTRING(1025)
                        END

ITVersionNameQueue      QUEUE,TYPE
ValueName                 CSTRING(50)
UsedName                  CSTRING(50)
                        END
```

3.36.2.1 ITVersionInfoQueue

Enter topic text here.

```
ITVersionInfoQueue      QUEUE,TYPE
ValueName                 CSTRING(50)
ValueUseName              CSTRING(50)
ValueInfo                 CSTRING(1025)
                        END
```

3.36.2.2 ITVersionNameQueue

Enter topic text here.

```
ITVersionNameQueue      QUEUE,TYPE
ValueName                 CSTRING(50)
UsedName                  CSTRING(50) ! Formatted name
                        END
```

3.36.3 Properties

Enter topic text here.

3.36.3.1 FileExists

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.36.3.2 FileHasVersionInfo

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.36.3.3 HideDebugViewVC

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.36.3.4 VersionInfo
Version Class - Properties

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.36.3.5 VersionNames
Version Class - Properties

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.36.4 Methods
Version Class

Enter topic text here.

3.36.4.1 AddClarionResources
Version Class - Methods

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.2 AddVersionName
Version Class - Methods

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.3 GetDisplayName

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.4 GetLanguageString

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.5 GetVersionInfo

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.6 LoadVersionNames

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.36.4.7 PTD

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.36.4.8 QueryValue

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.9 RetrieveFromFile

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.10 RetrieveFromSelf

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.36.4.11 Construct

Version Class - Methods

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.36.4.12 Destruct

Version Class - Methods

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

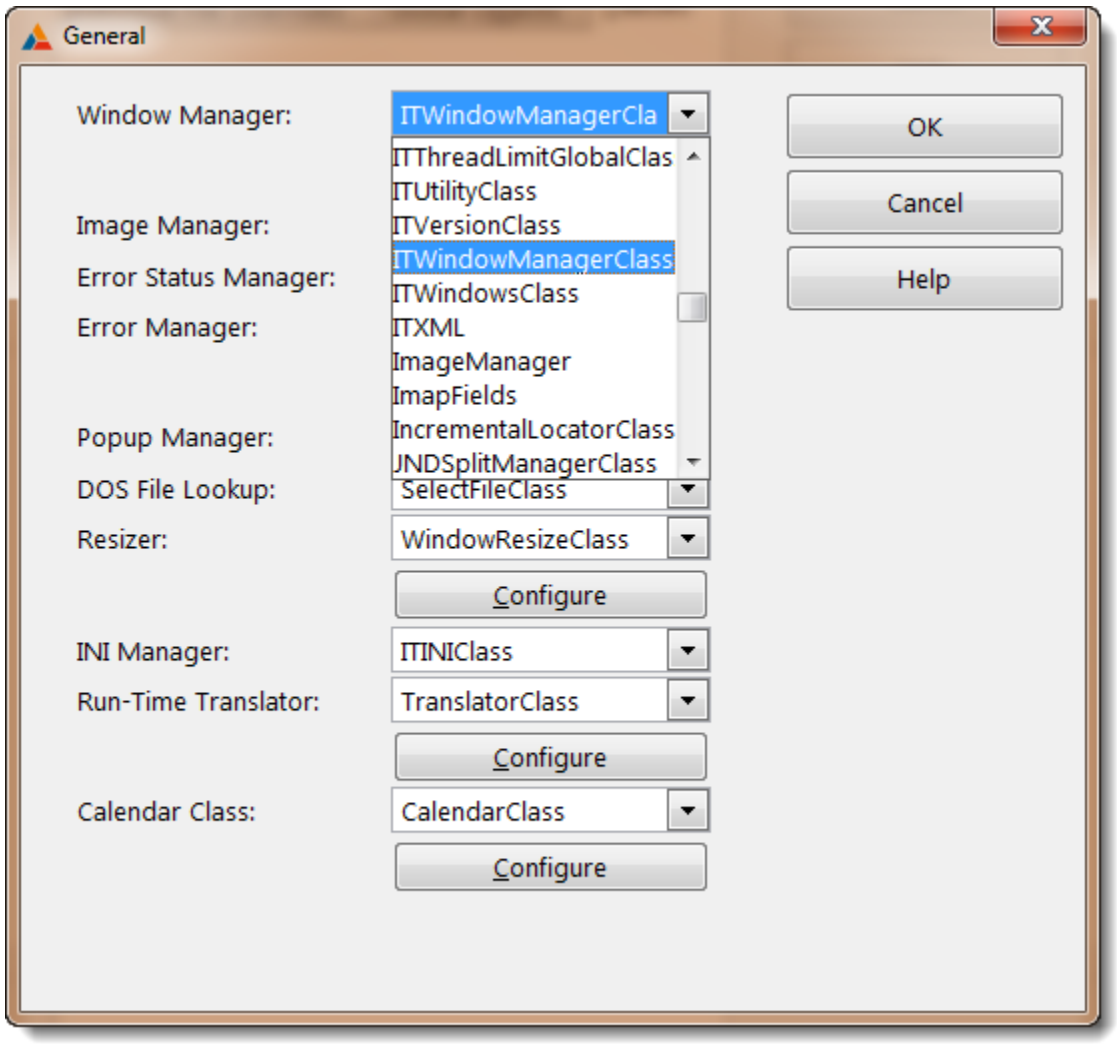
Example:

See also:

3.37 Window Manager Class

3.37.1 Overview Window Manager Class

This class is derrived from the ABC WindowManager class and can replace it to implement the Global Thread class 168 to the application. For it to work it is neccessary to change the Window Manger from WindowManager class to ITWindowManagerClass in the application global actions:



Once this is done you can use the Global Thread class to control closing windows and closing all threaded windows. Every window on a thread is added and is closed in reversed order of opening. For example if Proc1 calls Proc2 which calls Proc3, then Proc3 is first closed, then Proc2 and finally Proc1. Note that this cannot force the windows to close. If for example a form is open and changes have been made to a record it will not be able to close the form window.

```
ITWindowManagerClass
CLASS(WindowManager),TYPE,Module('ITWindowManagerClass.clw'),Link('ITWindowManagerC
lass',_ITUtilLinkMode_),DLL(_ITUtilDllMode_)
ThreadClass                &ITGlobalThreadClass
ErrorClass                 &ErrorClass
```

```
WindowRef                &Window

Init                     Procedure(Window pWIN, ITGlobalThreadClass pThreadClass,
ErrorClass pErrorClass)
Kill                     PROCEDURE,PROC,BYTE,VIRTUAL    ! Level:Notify means dead
already
TakeWindowEvent          PROCEDURE,VIRTUAL,BYTE,PROC
                       END
```

3.37.2 Properties — Window Manager Class

Enter topic text here.

3.37.2.1 ThreadClass — Window Manager Class - Properties

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.37.2.2 ErrorClass — Window Manager Class - Properties

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.37.2.3 WindowRef — Window Manager Class - Properties

The <property> property is used in the <methods used in> methods and is used to ...

It is declared as:

```
<property declaration>
```

3.37.3 Methods — Window Manager Class

Enter topic text here.

3.37.3.1 Init — Window Manager Class - Methods

Prototype:

pParameter Parameter Information

Returns Return information

Method information

Example:

See also:

3.37.3.2 Kill

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.37.3.3 TakeWindowEvent

Prototype:

pParameter	Parameter Information
Returns	Return information

Method information

Example:

See also:

3.38 Windows Class

3.38.1 Overview Windows Class

The Windows Class is the second class in the hierarchy and is derived from the Core Class [52]. The Windows class contains various useful functions that call on various core API functions.

```
ITWindowsClass
Class(ITCoreClass),TYPE,Module('ITWindowsClass.clw'),Link('ITWindowsClass',_ITUtilL
inkMode_),DLL(_ITUtilDllMode_)

AppframeClientHandle [368]      Long
ChildWindows [368]              &ChildWindowQ [367]
FrameColor [376]                Long
IsVista [369]                   Byte !! True if MajorVersion => 6
IsWindowOnTop [369]             Byte
MajorVersion [369]              Long
MinorVersion [370]              Long
ModuleWindows [371]             &ChildWindowQ [367]
SaveNewBrush [372]              Long
SaveOldBrush [372]              Long
ThemedControls [372]            &tThemedControls [367]
TopWindows [372]                &ChildWindowQ [367]
VersionBuildNr [373]            Long
VersionInformation [374]        String(128)
VistaHasUAC [375]               Byte
VersionPlatformID [374]         Long
W95HiBuildNr [376]              Short
W95LoBuildNr [376]              Short
WindowStyle [376]               Long
WindowsColorChanged [376]       Byte
LastActiveTime [377]            Long
LastActiveTick [377]            Long


ActivateWindow [378]            Procedure(String pWindowTitle)
APIErrorHandler [379]           Procedure(String pCaption),VIRTUAL
Disable64bitRedirection [379]   Procedure(),Byte,PROC  !! Disable 64bit File System
redirection.
EnumChildWin [380]              Procedure(Long phWnd),Long  ! Returns the number of
enumerated child windows
EnumModuleWin [380]             Procedure(String pModuleName),Long
EnumTopWin [381]                Procedure(),Long  ! Returns the number of enumerated
top windows
FindWindow [382]                Procedure(String pCaption, Byte pFindInCaption=True,
Byte pFullMatch=False),Long,PROC,VIRTUAL  ! Returns hwnd or 0
GetBaseControlName [383]        Procedure(Long pFEQ),String
GetBaseControlName [383]        Procedure(String pLabel, Byte pUpper),String
GetCommandLineLen [384]         Procedure(),Long
GetControlName [385]            Procedure(Long pFEQ, Byte pRemoveQM=0),String
GetDialogUnit [385]             Procedure(Byte pVertical),Long
GetExeFromWindowHandle [386]    Procedure(Long pHwnd),String
GetIdleTime [386]               Procedure(),LONG
GetLastInputTime [387]          Procedure(),LONG
GetPIDFromWindowHandle [387]    Procedure(Long pHwnd),IT_DWORD
GetPixelHeight [388]            Procedure(Long pFEQ),Long
GetPixelPos [389]               Procedure(Long pFEQ, Long pC),Long
GetPixelPosition [389]          Procedure(Long pFEQ,<*Long pX>,<*Long pY>,<*Long
```

```
pW>,<*Long pH>)
GetPixelWidth 390              Procedure(Long pFEQ),Long
GetPixelXPos 390               Procedure(Long pFEQ),Long
GetPixelYPos 391               Procedure(Long pFEQ),Long
GetPopupXY 391                 Procedure(Long pFEQ,<*Long pX>,<*Long pY>)
GetScreenBaseDPIRatio 392      Procedure(),Real
GetScreenDPI 392               Procedure(),Long
GetScreenDPIRatio 393          Procedure(),Real
GetScreenX 393                 Procedure(Long pFEQ),Long
GetScreenY 393                 Procedure(Long pFEQ),Long
GetSysMetrics 394              Procedure(Long pIndex),Long  ! Wrapper for
GetSystemMetrics
GetSysParamInfo 394            Procedure(Long pAction, <Long pParam>, <*? plpvParam>,
Long pWinIni),Long,PROC  !! Wrapper for SystemParametersInfo
GetTaskbarHeight 395           Procedure(),Long
GetThemedPanelFEQ 395          Procedure(Long pPanelFEQ),LONG
GetWindowVersion 396           Procedure(),String,PROC
IsProgramRunning 398           Procedure(String SemaphoreValue),BYTE
IsTerminalServer 398           Procedure(),Byte
MakeLangID 399                 Procedure(UShort usPrimaryLanguage, UShort
usSubLanguage),UShort
PlaceControlForDPI 400         Procedure(Long pFEQ, <Long pDesignDPI>)  ! Sets X, Y,
W and H
RedrawClientArea 400           Procedure
RemoveWindowColor 401          Procedure(),BYTE
ResizeControlForDPI 401        Procedure(Long pFEQ, <Long pDesignDPI>)  ! Sets W and
H
Revert64bitRedirection 402     Procedure(),Byte,PROC  !! Revert 64bit File System
redirection.
SetControlFonts 402            Procedure(Long pFrom, Long pTo)
SetControlPositions 402        Procedure(Long pFrom, Long pTo)
SetControlProp 403             Procedure(Long pFEQ, Long pProperty, String pValue)
SetPixelHeight 403             Procedure(Long pFEQ, Long pValue)
SetPixelPos 404                Procedure(Long pFEQ, Long pC, Long pValue)
SetPixelPosition 405           Procedure(Long pFEQ,<Long pX>,<Long pY>,<Long
pW>,<Long pH>)
SetPixelWidth 405              Procedure(Long pFEQ, Long pValue)
SetPixelXPos 406               Procedure(Long pFEQ, Long pValue)
SetPixelYPos 406               Procedure(Long pFEQ, Long pValue)
SetToolboxCaption 407          Procedure(Long phwnd, Byte pSetOn=True)
SetWindowColor 408             Procedure(Long pColor)
SetWindowNotOnTop 408          Procedure
SetWindowOnTop 408             Procedure
SetWindowPosition 409          Procedure(Long pX, Long pY, Byte pSetPixels=True)
SetWindowSize 409              Procedure(Long pWidth, Long pHeight, Byte
pSetPixels=True)
ThemeAPanel 410                Procedure(Long pPanelFEQ)
UsesClearType 410              Procedure(),Byte  !! Returns true/false if the
computer is using ClearType
UsingLargeFonts 410            Procedure(),Byte
WindowInfoToODS 411            Procedure(String pProcedureName),VIRTUAL

Construct 412                  Procedure
Destruct 412                   Procedure
                          End
```

For examples of how to use the Window class, please refer to the Example Program 514 procedures for the Windows Class 365.

See also:

Windows Functions on MSDN

3.38.2 Data Types — Windows Class

The Windows class uses one special data type for child window enumeration.

ChildWindowQ 367

See also:
ChildWindows 368
EnumChildWin 380
EnumChildWindowsProc 413

3.38.2.1 tThemedControls — Windows Class - Data Types

The **tThemedControls** queue is used to keep track of controls that have been themed with the ThemeAPanel 410 method. Use the GetThemedPanelFEQ 395 to retrieve the original panel FEQ.

```
tThemedControls    QUEUE,PRE(tThemeControls),TYPE
OriginalFEQ          Long
NewFEQ               Long
                   End
```

See also:
ThemeAPanel 410
GetThemedPanelFEQ 395

3.38.2.2 ChildWindowQ — Windows Class - Data Types

The **ChildWindowQ** is used to store the child window enumeration information, such as the handle, style, extra style bits and the window caption (text).

```
ChildWindowQ              QUEUE,TYPE
CwHwnd                      Long
Style                       Long
ExStyle                     Long
Text                        CString(1025)
UpperText                   CString(1025)
ModuleName                  CString(1025)
ModuleEXEName               CString(101)  ! Uppercased name of the EXE
ProcessID                   IT_DWORD
ClassName                   CString(1025) !! AB 2010-09-07:  Added to
retrive the classname for the window
                          END
```

The Text contains the caption text of the window. The UpperText contains the same text all upper case. The ModuleName contains the SHORTPATH of the module which the window belongs to. Note that the ModuleName is ONLY filled in the EnumTopWin 381, not in EnumChildWin 380 since all child windows of a process will belong to the main process. ClassName field was added on September 7, 2010 to more easily identify classes.

See also:
ChildWindows 368
EnumChildWin 380
EnumTopWin 381
EnumChildWindowsProc 413
EnumTopWindowsProc 412

3.38.3 Properties — Windows Class

The Windows class currently has 19 properties.

AppframeClientHandle 368	Long
ChildWindows 368	&ChildWindowQ 367
FrameColor 376	Long
IsVista 369	Byte !! True if MajorVersion => 6
IsWindowOnTop 369	Byte
MajorVersion 369	Long
MinorVersion 370	Long
ModuleWindows 371	&ChildWindowQ 367
SaveNewBrush 372	Long
SaveOldBrush 372	Long
ThemedControls 372	&tThemedControls 367
TopWindows 372	&ChildWindowQ 367
VersionBuildNr 373	Long
VersionInformation 374	String(128)
VistaHasUAC 375	Byte
VersionPlatformID 374	Long
W95HiBuildNr 376	Short
W95LoBuildNr 376	Short
WindowStyle 376	Long

3.38.3.1 AppframeClientHandle — Windows Class - Properties

This property is used in the SetWindowColor 408 and RemoveWindowColor 401 methods to store the window client handle:

```
SELF.AppframeClientHandle = 0{Prop:ClientHandle}
```

The RemoveWindowColor 401 is called automatically when EVENT:CloseWindow is fired.

See also:
SetWindowColor 408
RemoveWindowColor 401

3.38.3.2 ChildWindows — Windows Class - Properties

ChildWindows is a queue of type ChildWindowQ 367 that is used in the EnumChildWin 380 method to store information about the enumerated child windows.

```
ChildWindows                &ChildWindowQ ! Reference to a Child Windows
Queue
```

See also:
ChildWindowQ 367
EnumChildWin 380
EnumChildWindowsProc 413

3.38.3.3 IsVista — Windows Class - Properties

This property is set in the Constructor 412 method:

```
SELF.IsVista                = Choose(SELF.MajorVersion=>6,True,False)
```

This property is true for Vista and Windows Server 2008 For more information about the windows versions, please see http://msdn.microsoft.com/en-us/library/ms724451(VS.85).aspx and http://msdn.microsoft.com/en-us/library/ms724833(VS.85).aspx

See also:
Constructor 412
GetWindowVersion 396

3.38.3.4 IsWindowOnTop — Windows Class - Properties

This property is set to true or false in the SetWindowOnTop and SetWindowNotOnTop methods. It's value indicates if the window is set to be at the top of the z-order of windows. This means that the window will be on top of all other windows.

Example:

```
ITW  ITWindowsClass
 Code
 ! ...
 If Not ITW.IsWindowOnTop
   ITW.SetWindowOnTop
 End
```

See also:
SetWindowOnTop 408
SetWindowNotOnTop 408

3.38.3.5 MajorVersion — Windows Class - Properties

This property is a windows version property and indicates the major version number. For example 5 is the version number of Windows XP.

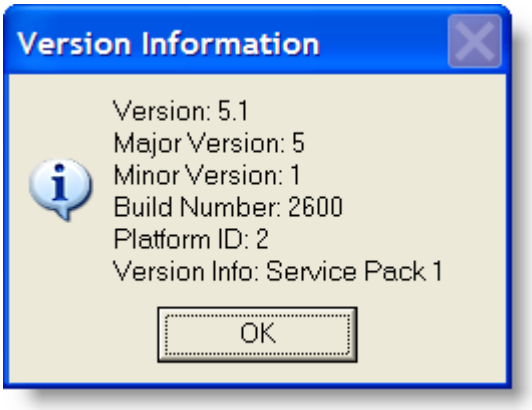
Value	Meaning

4	Windows NT 4.0, Windows Me, Windows 98, or Windows 95
5	Windows Server 2003 R2, Windows Server 2003, Windows XP, or Windows 2000
6	Windows Vista or Windows Server "Longhorn"

Example:

```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
        '|Major Version:     ' & ITW.MajorVersion &|
        '|Minor Version:     ' & ITW.MinorVersion &|
        '|Build Number:      ' & ITW.VersionBuildNr &|
        '|Platform ID:       ' & ITW.VersionPlatformID &|
        '|Version Info:      ' & ITW.VersionInformation, |
        'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:

GetWindowVersion 396
MinorVersion 370
VersionBuildNr 373
VersionPlatformID 374
VersionInformation 374

3.38.3.6 MinorVersion

Windows Class - Properties

This property is a windows version property and indicates the minor version number. For example 5 is the major version number of Windows XP and 1 is the minor version number of Service Pack 1.

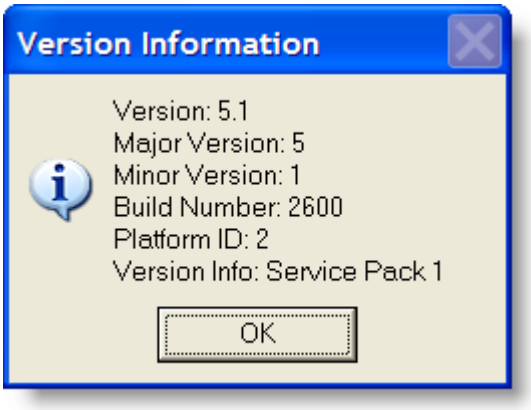
Value	Meaning
0	Windows Vista, Windows Server "Longhorn", Windows 2000, Windows NT 4.0, or Windows 95
1	Windows XP
2	Windows Server 2003 R2, Windows Server 2003, or Windows XP Professional x64 Edition
10	Windows 98

90	Windows ME

Example:

```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
         '|Major Version:    ' & ITW.MajorVersion &|
         '|Minor Version:    ' & ITW.MinorVersion &|
         '|Build Number:     ' & ITW.VersionBuildNr &|
         '|Platform ID:      ' & ITW.VersionPlatformID &|
         '|Version Info:     ' & ITW.VersionInformation, |
         'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:
GetWindowVersion 396
MajorVersion 369
VersionBuildNr 373
VersionPlatformID 374
VersionInformation 374

3.38.3.7 ModuleWindows — Windows Class - Properties

ModuleWindows is a queue of type ChildWindowQ 367 that is used in the EnumModuleWin 380 method to store information about the enumerated top windows for a specific module. By TopWindows we mean windows that are defined as Top-Level windows. Unlike child windows, top windows do not have parent windows. For a Clarion application this would be the appframe window. ChildWindows 368 would be any windows opened by that appframe window.

```
ChildWindows                &ChildWindowQ ! Reference to a Child Windows
Queue
```

See also:
EnumTopWin 381
EnumTopWindowsProc 412

3.38.3.8 SaveNewBrush

Windows Class - Properties

This property is set in the SetWindowColor 408 method and used in the RemoveWindowColor 401 method. It saves the new brush created for the window, which is then restored in RemoveWindowColor 401.

```
SELF.SaveNewBrush = IT_CreateSolidBrush(SELF.FrameColor)
```

See also:
SetWindowColor 408
RemoveWindowColor 401

3.38.3.9 SaveOldBrush

Windows Class - Properties

This property is set in the SetWindowColor 408 method and used in the RemoveWindowColor 401 method. It saves the original brush used for the window, which is then restored in RemoveWindowColor 401.

```
SELF.SaveOldBrush = IT_GetClassLong(SELF.AppframeClientHandle,
IT_GCL_HBRBACKGROUND)
```

See also:
SetWindowColor 408
RemoveWindowColor 401

3.38.3.10 ThemedControls

Windows Class - Properties

ThemedControls is a queue that is created with the Constructor 412 and added to by the ThemeAPanel 410 method. It contains the FEQ of the panel being themed and the tab Sheet that is created to replace the panel. It is used by the GetThemedPanelFEQ 395 to get the Sheet FEQ that matches the panel FEQ that was replaced. If XPThemes 57 are not present or the window is not themed, both the OriginalFEQ and the NewFEQ will be the same and equal to the Panel FEQ.

```
Add(SELF.ThemedControls,SELF.ThemedControls.OriginalFEQ)
```

See also:
Constructor 412
GetThemedPanelFEQ 395
ThemeAPanel 410

3.38.3.11 TopWindows

Windows Class - Properties

TopWindows is a queue of type ChildWindowQ 367 that is used in the EnumTopWin 381 method to store information about the enumerated top windows. By TopWindows we mean windows that are

defined as Top-Level windows. Unlike child windows, top windows do not have parent windows. For a Clarion application this would be the appframe window. ChildWindows 368 would be any windows opened by that appframe window.

```
ChildWindows                &ChildWindowQ ! Reference to a Child Windows
Queue
```

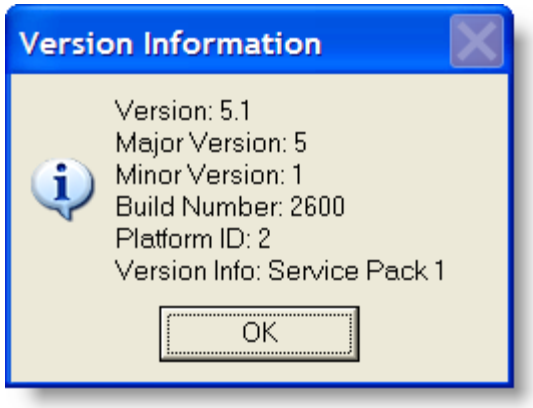
See also:
ChildWindowQ 367
EnumTopWin 381
EnumTopWindowsProc 412

3.38.3.12 VersionBuildNr — Windows Class - Properties

This property is a windows version property and indicates the build number. On Windows 95/98/ME, the low-order word 376 contains the build number of the operating system. The high-order word 376 contains the major and minor version numbers. On other versions, this contains a build number.

Example:
```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
         '|Major Version:    ' & ITW.MajorVersion &|
         '|Minor Version:    ' & ITW.MinorVersion &|
         '|Build Number:     ' & ITW.VersionBuildNr &|
         '|Platform ID:      ' & ITW.VersionPlatformID &|
         '|Version Info:     ' & ITW.VersionInformation, |
         'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:
GetWindowVersion 396
MajorVersion 369
MinorVersion 370
VersionPlatformID 374
VersionInformation 374

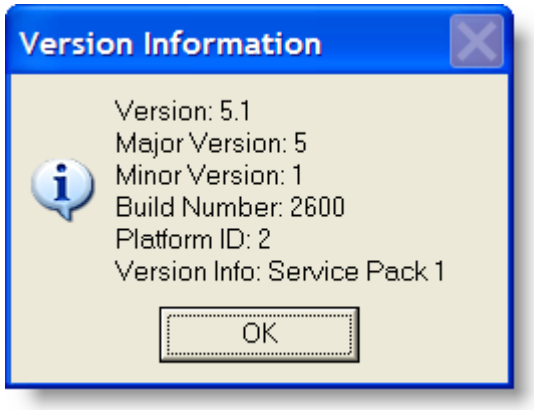
3.38.3.13 VersionInformation

This property is a windows version property and indicates additional version information. This can indicate a service pack or on Windows 95/98/ME it can be additional version information.

Example:

```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
        '|Major Version:     ' & ITW.MajorVersion &|
        '|Minor Version:     ' & ITW.MinorVersion &|
        '|Build Number:      ' & ITW.VersionBuildNr &|
        '|Platform ID:       ' & ITW.VersionPlatformID &|
        '|Version Info:      ' & ITW.VersionInformation, |
        'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:

GetWindowVersion 396
MajorVersion 369
MinorVersion 370
VersionPlatformID 374
VersionBuildNr 373

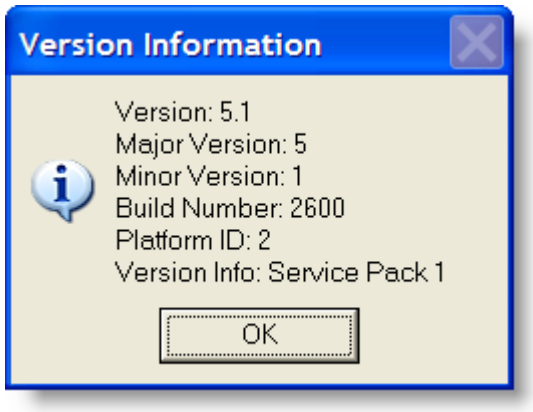
3.38.3.14 VersionPlatformID

This property is a windows version property and indicates version platform ID. This is either 1 (IT_VER_PLATFORM_WIN32_WINDOWS)
or 2 (IT_VER_PLATFORM_WIN32_NT), i.e. 1 indicates Windows 95/98/ME and 2 indicates Windows Vista, Windows Server "Longhorn", Windows Server 2003, Windows XP, Windows 2000, or Windows NT.

Example:

```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
         '|Major Version:    ' & ITW.MajorVersion &|
         '|Minor Version:    ' & ITW.MinorVersion &|
         '|Build Number:     ' & ITW.VersionBuildNr &|
         '|Platform ID:      ' & ITW.VersionPlatformID &|
         '|Version Info:     ' & ITW.VersionInformation, |
         'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:
GetWindowVersion 396
MajorVersion 369
MinorVersion 370
VersionPlatformID 374
VersionBuildNr 373

3.38.3.15 VistaHasUAC

Windows Class - Properties

This property will tell is UAC is turned on. To our knowledge this is the best way to determine if the UAC is on or not.

```
SELF.VistaHasUAC = GetReg(REG_LOCAL_MACHINE,
'Software\Microsoft\Windows\CurrentVersion\Policies\System','EnableLUA')
```

See also:
IsVista 369

3.38.3.16 W95HiBuildNr

This contains the High-order word of the VersionBuildNr 373. Currently this value is not set.

See also:
VersionBuildNr 373

3.38.3.17 W95LoBuildNr

This contains the Low-order word of the VersionBuildNr 373. Currently this value is not set.

See also:
VersionBuildNr 373

3.38.3.18 WindowColor

WindowColor is set in the SetWindowColor 408 method:

```
SELF.WindowColor = pColor
```

It is currently only used in that method.

See also:
SetWindowColor 408

3.38.3.19 WindowStyle

This is used in SetToolboxCaption 407 to store the Window Style value as returned by the GetWindowLong api call. This is stored in order to be able to change the caption bar to toolbox caption and then later change it back with all the previous styles intact.

See also:
SetToolboxCaption 407

3.38.3.20 WindowsColorChanged

WindowsColorChanged is set in SetWindowColor 408 and is also used in RemoveWindowColor 401. It is set to either True or False depnding on if the background color for the window has been changed using SetWindowColor 408.

See also:
SetWindowColor 408
RemoveWindowColor 401

3.38.3.21 LastActiveTime — Windows Class - Properties

The LastActiveTime property is used in the GetIdleTime 386 method and is used to keep track of the last time the system was active by user input.

It is declared as:

```
LastActiveTime          Long
```

3.38.3.22 LastActiveTick — Windows Class - Properties

The LastActiveTick property is used in the GetIdleTime 386 method and is used to keep track of the millisecond ticker counter. If it doesn't change, the system is idle. If it changes the system is not idle.

It is declared as:

```
LastActiveTick          Long
```

3.38.4 Methods — Windows Class

The Windows class has 54 methods, including the Constructor and Destructor.

```
APIErrorHandler 379          Procedure(String pCaption),VIRTUAL
EnumChildWin 380             Procedure(Long phWnd),Long  ! Returns the number of
enumerated child windows
EnumModuleWin 380            Procedure(String pModuleName),Long
EnumTopWin 381               Procedure(),Long  ! Returns the number of enumerated
top windows
FindWindow 382               Procedure(String pCaption, Byte pFindInCaption=True,
Byte pFullMatch=False),Long,PROC,VIRTUAL  ! Returns hwnd or 0
GetBaseControlName 383       Procedure(Long pFEQ),String
GetBaseControlName 383       Procedure(String pLabel, Byte pUpper),String
GetCommandLineLen 384        Procedure(),Long
GetControlName 385           Procedure(Long pFEQ, Byte pRemoveQM=0),String
GetDialogUnit 385            Procedure(Byte pVertical),Long
GetExeFromWindowHandle 386   Procedure(Long pHwnd),String
GetPIDFromWindowHandle 387   Procedure(Long pHwnd),IT_DWORD
GetPixelHeight 388           Procedure(Long pFEQ),Long
GetPixelPos 389              Procedure(Long pFEQ, Long pC),Long
GetPixelPosition 389         Procedure(Long pFEQ,<*Long pX>,<*Long pY>,<*Long
pW>,<*Long pH>)
GetPixelWidth 390            Procedure(Long pFEQ),Long
GetPixelXPos 390             Procedure(Long pFEQ),Long
GetPixelYPos 391             Procedure(Long pFEQ),Long
GetPopupXY 391               Procedure(Long pFEQ,<*Long pX>,<*Long pY>)
GetScreenBaseDPIRatio 392    Procedure(),Real
GetScreenDPI 392             Procedure(),Long
GetScreenDPIRatio 393        Procedure(),Real
GetScreenX 393               Procedure(Long pFEQ),Long
GetScreenY 393               Procedure(Long pFEQ),Long
GetSysMetrics 394            Procedure(Long pIndex),Long  ! Wrapper for
GetSystemMetrics
GetTaskbarHeight 395         Procedure(),Long
```

```
GetThemedPanelFEQ 395          Procedure(Long pPanelFEQ),LONG
GetWindowVersion 396           Procedure(),String,PROC
MakeLangID 399                 Procedure(UShort usPrimaryLanguage, UShort
usSubLanguage),UShort
PlaceControlForDPI 400         Procedure(Long pFEQ, <Long pDesignDPI>)  ! Sets X, Y,
W and H
RedrawClientArea 400           Procedure
RemoveWindowColor 401          Procedure(),BYTE
ResizeControlForDPI 401        Procedure(Long pFEQ, <Long pDesignDPI>)  ! Sets W and
H
SetControlFonts 402            Procedure(Long pFrom, Long pTo)
SetControlPositions 402        Procedure(Long pFrom, Long pTo)
SetControlProp 403             Procedure(Long pFEQ, Long pProperty, String pValue)
SetPixelHeight 403             Procedure(Long pFEQ, Long pValue)
SetPixelPos 404                Procedure(Long pFEQ, Long pC, Long pValue)
SetPixelPosition 405           Procedure(Long pFEQ,<Long pX>,<Long pY>,<Long
pW>,<Long pH>)
SetPixelWidth 405              Procedure(Long pFEQ, Long pValue)
SetPixelXPos 406               Procedure(Long pFEQ, Long pValue)
SetPixelYPos 406               Procedure(Long pFEQ, Long pValue)
SetToolboxCaption 407          Procedure(Long phwnd, Byte pSetOn=True)
SetWindowColor 408             Procedure(Long pColor)
SetWindowNotOnTop 408          Procedure
SetWindowOnTop 408             Procedure
SetWindowPosition 409          Procedure(Long pX, Long pY, Byte pSetPixels=True)
SetWindowSize 409              Procedure(Long pWidth, Long pHeight, Byte
pSetPixels=True)
ThemeAPanel 410                Procedure(Long pPanelFEQ)
UsesClearType 410              Procedure(),Byte  !! Returns true/false if the
computer is using ClearType
UsingLargeFonts 410            Procedure(),Byte
WindowInfoToODS 411            Procedure(String pProcedureName),VIRTUAL

Construct 412                  Procedure
Destruct 412                   Procedure
```

3.38.4.1 ActivateWindow

Prototype: **(String pWindowTitle, Long pWindowHandle=0)**

pWindowTitle	Full or partial title of the window to activate.
pWindowHandle	Optionally pass in the handle of the window to activate. (New in build 1.2.2427, November 24, 2014)

This method is used to activate a window. It is used by the Limit Program Instance 454 template to activate the window for the program being limited. It can be used with any parent or top window and will bring it into focus and restore it if it is minimized.

Example:

```
Glo:ITWindowsClass   ITWindowsClass                    ! Icetips: Limit
Program Instance
  CODE
  If Glo:ITWindowsClass.IsProgramRunning('This is a test') ! Icetips: Limit
Program Instance
    Message('The program is already running.','Program is already running')
! Icetips: Limit Program Instance
```

```
    Glo:ITWindowsClass.ActivateWindow('Registry Class test program') !
Icetips: Limit Program Instance
    HALT()                                                  ! Icetips: Limit
Program Instance
  End                                                       ! Icetips: Limit
Program Instance
```

See also:
IsProgramRunning 398

Limit Program Instance 454

3.38.4.2 APIErrorHandler — Windows Class - Methods

Prototype: **(String pCaption),VIRTUAL**

pCaption Message Caption

This is a placeholder virtual method that is used in the ShellClass 257 and can be used anywhere to create a special API error handler if needed.

3.38.4.3 Disable64bitRedirection — Windows Class - Methods

Prototype: **(),Byte,PROC**

Returns TRUE if the method was succesfull in enabling 64bit redirection

This method is used to disable 64bit redirection for registry and file system so 32bit programs can access 64bit registry key and filenames. Use Revert64bitRedirection 402 to set it back to normal 32bit access.

Example:

```
ITS          ITShellClass
FileName     CSTRING(1025)
ProgramFile  CSTRING(1025)
  CODE
  If FileName
    If ITS.Disable64bitRedirection()
      ProgramFile = ITS.GetAssociatedProg(FileName)
      If Not ITS.Revert64bitRedirection()
        Message('Could not change redirection!')
        Stop
      End
    End
```

See also:
Revert64bitRedirection 402

3.38.4.4 EnumChildWin

Prototype: **(Long phWnd),Long**

phWnd Window handle to enumerate child windows for, i.e. parent window

Returns Number of child windows

This method is used to enumerate all child windows of a parent window specified in the phWnd parameter. Please note that this does not enumerate non-MDI windows that have been opened by the appframe if the appframe handle is the phWnd passed to this method. It will only enumerate MDI windows. Also note that all controls are considered child windows.

Example:

```
ITW  ITWindowsClass
 Code

SetupWindow          ROUTINE
 Data
I  Long
 Code
 If ITW.EnumChildWin(pParentHandle)
   Loop I = 1 To Records(ITW.ChildWindows)
     Get(ITW.ChildWindows,I)
     ChildWindows.CW:CwHwnd   = ITW.ChildWindows.CwHwnd
     ChildWindows.CW:Style    = ITW.ChildWindows.Style
     ChildWindows.CW:ExStyle  = ITW.ChildWindows.ExStyle
     ChildWindows.CW:Text     = ITW.ChildWindows.Text
     Add(ChildWindows)
   End
 End
```

Please refer to the Example Program 514 TestEnumChildWindows 514 procedure for examples of implementation.

See also:
EnumTopWin 381

3.38.4.5 EnumModuleWin

Prototype: **(String pModuleName)!!,Long**

pModuleName Name of the EXE file to enumerate top windows for. This must be in the form of FileName+Extension, such as NOTEPAD.EXE. Note that this parameter is converted to LongPath() for compatibility with the ModuleEXEName 367 member of the ChildWindowQ 367 queue type.

Returns Returns the number of windows enumerated for the module.

Beta 3.3: This method has not been tested thoroughly yet and should be used carefully!

This method can be used to enumerate all windows that are related to a specified executable module, specified in the pModuleName parameter. This method uses the EnumTopWin method to enumerate all top windows for the module.

Example:

```
ITW  ITWindowsClass
 Code
 If ITW.EnumModuleWin('EXPLORER.EXE')
 End
```

See also:
EnumTopWin 381
ModuleWindows 371
ChildWindowQ 367

3.38.4.6 EnumTopWin — Windows Class - Methods

Prototype: **(),Long**

Returns Number of top windows

This method is used to enumerate all top windows running on the machine. This can come in handy when searching for a window where only part of the caption is known, i.e. "EXCEL" or "WORD". The TopWindows queue contains both the caption in the Text variable and also the upper cased text in the UpperText variable.

Example:

```
ITW  ITWindowsClass
 Code

SetupWindow          ROUTINE
 Data
I  Long
 Code
 If ITW.EnumTopWin()
   Loop I = 1 To Records(ITW.TopWindows)
     Get(ITW.TopWindows,I)
     TopWindows.CW:CwHwnd   = ITW.TopWindows.CwHwnd
     TopWindows.CW:Style    = ITW.TopWindows.Style
     TopWindows.CW:ExStyle  = ITW.TopWindows.ExStyle
     TopWindows.CW:Text     = ITW.TopWindows.Text
     Add(TopWindows)
   End
 End
```

Here is an example that will only load windows based on a search criteria.

```
ITW  ITWindowsClass
 Code

FindTopWindows          ROUTINE
 Data
I  Long
FS CString(Size(Loc:Search)+1)
 Code
```

```
FS = Upper(Clip(Loc:Search))
Free(TopWindows)
If ITW.EnumTopWin()
  Loop I = 1 To Records(ITW.TopWindows)
    Get(ITW.TopWindows,I)
    If Instring(FS,ITW.TopWindows.UpperText,1,1)
      TopWindows.CW:CwHwnd   = ITW.TopWindows.CwHwnd
      TopWindows.CW:Style    = ITW.TopWindows.Style
      TopWindows.CW:ExStyle  = ITW.TopWindows.ExStyle
      TopWindows.CW:Text     = ITW.TopWindows.Text
      Add(TopWindows)
    End
  End
End
```

Please refer to the Example Program 514 TestEnumChildWindows 514 procedure for examples of implementation.

See also:
EnumChildWin 380

3.38.4.7 FindWindow

Prototype:	**(String pCaption, Byte pFindInCaption=True, Byte pFullMatch=False),Long,VIRTUAL**
pCaption	String to find
pFindInCaption	Indicates if the Caption or the EXE name should be searched. This parameter defaults to True
pFullMatch	If true the stirng is compared directly. If False the string is compared with Instring. This parameter defaults to False. **The search is ALWAYS case insensitive!**
Returns	Window handle if a window was found. If no window was found the return value is 0 (zero)

This method searches the caption text of all top windows using Instring. Note that the TopWindows property is loaded with the enumerated top windows after a call to this method, so you can perform more detailed search if the FindWindow returns zero. Please note that this will return the first window found and not indicate if there are more windows that fit the search criteria. For that type of code, please see the examples for EnumTopWin 381 and the code in the Example Program 514 for EnumTopWin 381.

NEW Beta 3.3

This method can now be used to search for a top window where the EXE name matches the pCaption parameter. The other new parameter can be used to specify exact search string, i.e. it will not find "NOTEPAD" in "NOTEPAD.EXE" if this parameter is set to TRUE. If the parameter is false it would find "NOTEPAD" in "NOTEPAD.EXE" Note that when you pass a executable name in pCaption, it is stripped to the filename and extension only and then compared to the ModuleEXEName which also contains just the filename and extension. If the pCaption contains a '\' then it is treated as a possible path and only the filename and extension are extracted from it.

PLEASE NOTE: This method simply returns the first window handle that matches. There may be multiple top windows for a given process. Our next release will have a method that enumerates all top windows for a given executable along with more information about each window that is enumerated.

Example:

```
ITW  ITWindowsClass
 Code

TestFindWindow              ROUTINE
 Data
hWnd Long
 Code
 hWnd = ITW.FindWindow(Loc:Search,Loc:SearchInCaption)
 If hWnd
   Message('Window found, handle returned = ' & hWnd,'Window found',ICON:
Exclamation)
 Else
   Message('Window not found','Window not found',ICON:Hand)
 End
```

If Loc:Search contains "Notepad.exe" and Loc:SearchInCaption is true, then this will search for a top window belonging to Notepad.exe.

See also:
EnumTopWin 381
EnumChildWin 380

3.38.4.8 GetBaseControlName — Windows Class - Methods

Prototype:	**(String pLabel, Byte pUpper),String**
pLabel	The Field EQuate label name.
pUpper	Indicates if the Base name should be uppercased before it is returned
Returns	String containing the field name

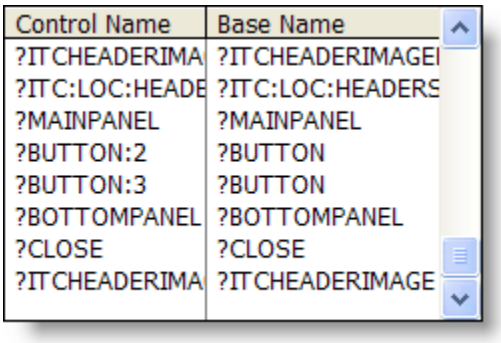
This method is called by the GetBaseControName function with the label of the control to get the base control name for. The base control name is for example ?INSERT for an control called ?INSERT:3 i.e. it strips any numbers off of the end so any control with that base name can be address for example when looping through all controls on a window or a report.

Example:

```
ITW  ITWindowsClass
 Code

LoadControlQ             ROUTINE
 Data
I  Long
 Code

 Loop I = FirstField() To LastField()
```

```
    Loc:CQ.Loc:CQName     = ITW.GetControlName(I)
    Loc:CQ.Loc:CQBaseName = ITW.GetBaseControlName(I)
    Add(Loc:CQ)
  End
```

Below is a screenshot of the Loc:CQ in a listbox. Please note the ?BUTTON:2 and ?BUTTON:3 in the Control Name column and also in the Base Name column. Also note that ?CLOSE has not changed.



Control Name	Base Name
?ITCHEADERIMA	?ITCHEADERIMAGEI
?ITC:LOC:HEADE	?ITC:LOC:HEADERS
?MAINPANEL	?MAINPANEL
?BUTTON:2	?BUTTON
?BUTTON:3	?BUTTON
?BOTTOMPANEL	?BOTTOMPANEL
?CLOSE	?CLOSE
?ITCHEADERIMA	?ITCHEADERIMAGE

See also:
GetControlName 385

3.38.4.9 GetCommandLineLen — Windows Class - Methods

Prototype: **(),Long**

Returns Returns the length of a command line that a program can accept

This method attempts to calculate the total length of a command line acceptable based on the operating system. This is not failsafe and should be used only as a guideline. Additional information about the maximum size of the command line can be found on Microsoft's Support website at http://support.microsoft.com/kb/830473 and on MSDN website at http://msdn2.microsoft.com/en-us/library/ms724834.aspx

This method can be used to determine if a constructed command line is too long for the operating system to pass it to another program. If it is too long and you are passing filenames to another program, consider using ShortPath() on the filename before placing it in the command line.

Example:

```
L   Long
ITW ITWindowsClass
 Code
 L = ITW.GetCommandLineLen()
 Message('Maximum length of the command line is ' & L & ' characters.')
```

3.38.4.10 GetControlName

Windows Class - Methods

Prototype: **(Long pFEQ),String**

pFEQ The Field EQuate label of a control to get the name for.

Returns Returns a string containing the label of the control.

This method uses an undocumented function in the Clarion Runtime Library that returns the label of a control, for example '?String1' or '?MYF:Field1' as a string rather than a numeric value.

Example:

```
ITW  ITWindowsClass
 Code

LoadControlQ            ROUTINE
 Data
I  Long
 Code

 Loop I = FirstField() To LastField()
   Loc:CQ.Loc:CQName     = ITW.GetControlName(I)
   Loc:CQ.Loc:CQBaseName = ITW.GetBaseControlName(I)
   Add(Loc:CQ)
 End
```

See also:
GetBaseControlName 383

3.38.4.11 GetDialogUnit

Windows Class - Methods

Prototype: **(Byte pVertical),Long**

pVertical Indicates if the Dialog unit value should be for Vertical or Horizontal.

Returns Returns the dialog base units for height or width of a standard system font character.

The Dialog Units are used in Clarion (and other languages) to get uniform sizes of dialogs independent on what fonts are used for the window and what resolution is used. This method returns the calculated values based on the GetDialogBaseUnits api. This method is used in the UsingLargeFonts 410 method to determine the size of the dialog units.

Example:

```
ITW  ITWindowsClass
 Code

 Loc:DialogUnitsString = 'Dialog units: ' &|
                         ITW.GetDialogUnit(False) & 'x' &|
                         ITW.GetDialogUnit(True)
```

See also:
UsingLargeFonts 410

3.38.4.12 GetExeFromWindowHandle

Prototype: **(Long pHwnd),String**

pHwnd Handle to a window.

Returns Returns EXE filename that owns the window passed to the method

This method returns the name of the process that owns the window handle that is passed to the method. This can be very useful if you know a handle but need to find the executable that it belongs to.

Example:

```
Mn  CString(2049)
ITW ITWindowsClass
 Code
 Mn = ITW.GetExeFromWindowHandle(0{Prop:Handle})
 Message('Owner of this window is: ' & Mn,'GetExeFromWindowHandle',ICON:
Exclamation)
```

See also:
EnumTopWin 381
EnumModuleWin 380
EnumChildWin 380
EnumChildWindowsProc 413
EnumTopWindowsProc 412
GetPIDFromWindowHandle 387

3.38.4.13 GetIdleTime

Prototype: **(),LONG**

Returns Returns a Clarion Time (i.e. 1/100 seconds) that have elapsed since the last user input has occurred.

This method can be used to easily monitor the time that the system has not been active, i.e. no user input registered. It uses the GetLastInputTime 387 to get the millisecond number when the last user input occurred and compares it to the value of the LastActiveTick 377 property. It then calculates the elapsed time by using the LastActiveTime 377 which is reset any time the method is called an the ticker has changed. It would generally be appropriate to call this method from a timer event handler

Example:

```
ITW  ITWindowsClass
Loc:IdleTime         TIME                                  !
```

```
W    WINDOW('Check Idle Time on system'),AT(,,194,138),FONT('Segoe UI',10
,,),TIMER(100),GRAY
       STRING(@t04),AT(63,31),USE(Loc:IdleTime),TRN,RIGHT,FONT(,18,,FONT:
bold)
       BUTTON('Close'),AT(75,106,45,14),USE(?Close)
     END

 Code
 Open(W)
 Accept
   Case EVENT()
   Of EVENT:Timer
     Loc:IdleTime = ITW.GetIdleTime()
     Display
   End
   If Field() = ?Close And Event() = EVENT:Accpted
     Break
   End
 End
 Close(W)
```

See also:
LastActiveTime 377
LastActiveTick 377
GetLastInputTime 387

3.38.4.14 GetLastInputTime — Windows Class - Methods

Prototype: **(),LONG**

Returns Returns the number of milliseconds since the last user input action occurred on the system.

This method is used by the GetIdleTime 386 method to find out how long the system has been idle. The method returns the number of ticks or milliseconds since the last user input occurred, i.e. either mouse movement/click or keyboard click. It does not consider program execution as user action, i.e. if a program is left running and it is performing some operation the ticker is not affected. As long as no mouse of keyboard input is registered the method returns the same number.

Example:

```
ITW  ITWindowsClass
T    Long
  Code
  T = ITW.GetLastInputTime()
```

See also:
GetIdleTime 386

3.38.4.15 GetPIDFromWindowHandle — Windows Class - Methods

Prototype: **(Long pHwnd),IT_DWORD**

pHwnd Handle to a window.

Returns Returns the Process ID (PID) for the process that owns the window.

This method returns the process ID of a window. It can be useful to identify what process a window belongs to in order to get process information.

Example:

```
ITW ITWindowsClass
PID  IT_DWORD
 Code
 PID = ITW.GetPIDFromWindowHandle(0{Prop:Handle})
 Message('Process ID of this window is: ' & PID,'GetPIDFromWindowHandle',
ICON:Exclamation)
```

See also:
EnumTopWin 381
EnumModuleWin 380
EnumChildWin 380
EnumChildWindowsProc 413
EnumTopWindowsProc 412
GetExeFromWindowHandle 386

3.38.4.16 GetPixelHeight

Windows Class - Methods

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label of the control to get the Height in pixels for.

Returns Returns the height of the control in pixels

This method returns the height of the passed control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Button height: ' & ITW.GetPixelHeight(?Button1))
```

See also:
GetPixelWidth 390
GetPixelXPos 390
GetPixelYPos 391

3.38.4.17 GetPixelPos

Windows Class - Methods

Prototype: **(Long pFEQ, Long pC),Long**

pFEQ The Field EQuate label for the control to get position in pixels for.

pC Property to use, this could be PROP:XPos, PROP:YPos, PROP:Width or PROP:Height

Returns Returns the appropriate property value in Pixels.

This method is used by GetPixelHeight 388, GetPixelWidth 390, GetPixelXPos 390 and GetPixelYPos 391 to get the pixel coordinates and size for the pFEQ control.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Button X: ' & ITW.GetPixelPos(?Button1,PROP:Xpos))
```

See also:
GetPixelHeight 388
GetPixelWidth 390
GetPixelXPos 390
GetPixelYPos 391

3.38.4.18 GetPixelPosition

Windows Class - Methods

Prototype: **(Long pFEQ,<*Long pX>,<*Long pY>,<*Long pW>,<*Long pH>)**

pFEQ The Field EQuate label for the control to get position in pixels for.
[pX] Optional parameter to receive the pFEQ contols X position in pixels.
[pY] Optional parameter to receive the pFEQ contols Y position in pixels.
[pW] Optional parameter to receive the pFEQ controls Width in pixels.
[pH] Optional parameter to receive the pFEQ controls Height in pixels.

This method can be used to retrieve one or more of a control's coordinates and size in pixels.

Example:

```
ITW  ITWindowsClass
 Code

GetButtonSizeInPixels  ROUTINE
 Data
X  Long
Y  Long
W  Long
H  Long
 Code
 ITW.GetPixelPosition(?Button,X,Y,W,H)
```

See also:
GetPixelPos 389

GetPixelHeight 388
GetPixelWidth 390
GetPixelXPos 390
GetPixelYPos 391

3.38.4.19 GetPixelWidth

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label of the control to get the Width in pixels for.

Returns Returns the width of the control in pixels

This method calls GetPixelPosition 389 and returns the Width of the pFEQ control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Button width: ' & ITW.GetPixelWidth(?Button1))
```

See also:
GetPixelHeight 388
GetPixelXPos 390
GetPixelYPos 391

3.38.4.20 GetPixelXPos

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label of the control to get the X position in pixels for.

Returns Returns the height of the control in pixels

This method calls GetPixelPosition 389 and returns the X position of the pFEQ control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Button Xpos: ' & ITW.GetPixelXpos(?Button1))
```

See also:
GetPixelHeight 388
GetPixelWidth 390
GetPixelYPos 391

3.38.4.21 GetPixelYPos

Windows Class - Methods

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label of the control to get the Y position in pixels for.

Returns Returns the height of the control in pixels

This method calls GetPixelPosition 389 and returns the Y position of the pFEQ control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Button Ypos: ' & ITW.GetPixelYpos(?Button1))
```

See also:
GetPixelHeight 388
GetPixelWidth 390
GetPixelXPos 390

3.38.4.22 GetPopupXY

Windows Class - Methods

Prototype: **(Long pFEQ,<*Long pX>,<*Long pY>)**

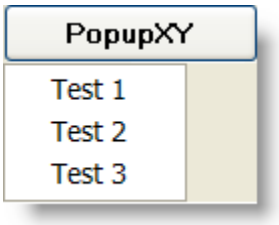
pFEQ The Field EQuate label for the control to get the Popup X and Y information for
[pX] Optional parameter to receive the X pixel position value
[pY] Optional parameter to receive the Y pixel position value

This method can be used to get the X and Y coordinates for the Clarion POPUP statement so the popup menu appears in an exact location relative to a specified control, passed in the pFEQ parameter. This is useful if you want a menu to appear below a button for example.

Example:

```
ShowPopupMenu            ROUTINE
 Data
X    Long
Y    Long
P    Short
 Code
 ITW.GetPopupXY(?PopupMenuButton,X,Y)
 Y += ITW.GetPixelHeight(?PopupMenuButton)
 P = Popup('Test 1|Test 2|Test 3',X,Y)
```

This results in a button with a popup menu below it as seen in the screenshot below:



See also:
GetPixelHeight 388

3.38.4.23 GetScreenBaseDPIRatio

Prototype: **(),Real**

Returns Ratio between 96 and the Screen DPI.

This method returns the ratio between a screen using 96DPI (normal font size) and the current screen DPI as returned by GetScreenDPI 392. This can be useful to scale controls **down** when designing on windows with large font settings. In order to scale **up**, use GetScreenDPIRatio 393.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Screen Base DPI ratio: ' & ITW.GetScreenBaseDPIRatio())
```

See also:
GetScreenDPI 392
GetScreenDPIRatio 393

3.38.4.24 GetScreenDPI

Prototype: **(),Long**

Returns Returns the value from GetDeviceCaps for LOGPIXELSX on the desktop window.

This method uses the GetDeviceCaps api to determine the number of pixels per logical inch along the screen width. This can be used to determine the size of a control in inches or other units as seen on the screen. For example if the DPI is 96 then a control that is 96 pixels wide should be exactly 1 inch wide or 25.4mm.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Screen DPI: ' & ITW.GetScreenDPI())
```

See also:
GetScreenBaseDPIRatio 392
GetScreenDPIRatio 393

3.38.4.25 GetScreenDPIRatio

Windows Class - Methods

Prototype: **(),Real**

Returns Ratio between 96 and the Screen DPI.

This method returns the ratio between a screen using 96DPI (normal font size) and the current screen DPI as returned by GetScreenDPI 392. This can be useful to scale controls **up** when designing on windows with large font settings. In order to scale **down**, use GetScreenBaseDPIRatio 392.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Screen DPI Ratio: ' & ITW.GetScreenDPIRatio())
```

See also:
GetScreenDPI 392
GetScreenBaseDPIRatio 392

3.38.4.26 GetScreenX

Windows Class - Methods

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label for the control to get the Screen X position for.

Returns Returns the screen X position in pixels for the control

This method returns the Screen X Position for the pFEQ control. This uses the ClientToScreen api to retrieve the screen coordinates for the control.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Screen X position: ' & ITW.GetScreenX(?Button1))
```

See also:
GetScreenY 393

3.38.4.27 GetScreenY

Windows Class - Methods

Prototype: **(Long pFEQ),Long**

pFEQ The Field EQuate label for the control to get the Screen Y position for.

Returns Returns the screen Y position in pixels for the control

This method returns the Screen Y Position for the pFEQ control. This uses the ClientToScreen api to retrieve the screen coordinates for the control.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Screen Y position: ' & ITW.GetScreenY(?Button1))
```

See also:
GetScreenX 393

3.38.4.28 GetSysMetrics

Prototype: **(Long pIndex),Long**

pIndex The Index value for GetSystemMetrics.

Returns Returns the value from GetSystemMetrics.

This method calls the GetSystemMetrics api directly. Note that most or all the index values are available in the ITUtilityClass. See the ITWin32Equates.inc for more information. Search for "!! GetSystemMetrics equates" in the file or IT_SM_ to get to the appropriate section in the file. This method simply calls GetSystemMetrics and returns the value returned from the api.

Example:

```
ITW  ITWindowsClass
 Code
 Message('Window Border Width: ' & ITW.GetSysMetrics(IT_SM_CXBORDER))
 Message('Window Caption Height: ' & ITW.GetSysMetrics(IT_SM_CYCAPTION))
```

See also:
GetSystemMetrics

3.38.4.29 GetSysParamInfo

Prototype: **(Long pAction, <Long pParam>, <*? plpvParam>, Long pWinIni),Long,PROC**

pAction A system wide parameter to be retrieved or set. See MSDN for more information. The equates for this are named IT_SPI_* in the Icetips Utilities. The IT_SPI equates are all defined in the ITWin32Equates.inc file.

pParam An additional parameter for the action to retrieve or set. It depends on the value passed to pAction. See MSDN for more information.

pplpvParam An additional parameter for the action to retrieve or set. It depends on the value passed to pAction. See MSDN for more information.

pWinIni Specifies if user profile should be updated. See MSDN for more information.

Returns If the method fails the return value is zero (0). If it succeeds the return value is not zero (0).

This method is a direct wrapper around the SystemParametersInfo api call. See MSDN for more information.

Example:

```
ITW  ITWindowsClass
Toggle IT_BOOL
 Code
 Toggle = false
 ITW.GetSysParamInfo(SPI_SETCLIENTAREAANIMATION, ,Toggle,
IT_SPIF_UPDATEINIFILE + IT_SPIF_SENDCHANGE)  !! Toggles animation on the
client area.
```

See also:

http://msdn.microsoft.com/en-us/library/ms724947(VS.85).aspx

3.38.4.30 GetTaskbarHeight

Windows Class - Methods

Prototype: **(),Long**

Returns Return the height of the Windows taskbar in pixels

This method uses the FindWindow and GetWindowRect api functions to get the height of the shell tray window and thus the taskbar. There are other ways to do this that can also take into account other dockable toolbars such as MS Office etc. and we may add those later if there is interest.

Example:

```
ITW ITWindowsClass
TBH  Long
 Code
 TBH = ITW.GetTaskBarHeight()
 Message('Toolbar Height: ' & TBH)
```

See also:
FindWindow
GetWindowRect

3.38.4.31 GetThemedPanelFEQ

Windows Class - Methods

Prototype: **(Long pPanelFEQ)**

pPanelFEQ The Field EQuate label of the original panel.

Returns Returns the Field EQuate label of the tab sheet that was created

This method can be used if you use ThemeAPanel 410 method to theme a panel using XPThemes 57. The ThemeAPanel 410 hides the panel and creates a wizard tabsheet instead creating an illusion of a themed panel! But if you want to change any settings on the "new" panel, i.e. the tabsheet, you need to use the GetThemedPanelFEQ to get the control FEQ.

Example:

```
ITW   ITWindowsClass
SFEQ  Long
PFEQ  Long
 Code
 PFEQ =  ?MainPanel
 SFEQ = ITW.GetThemedPanelFEQ(PFEQ)
 Message('Sheet FEQ = ' & SFEQ & '|Panel FEQ = ' & PFEQ &|
         '||Because the ?MainPanel is not themed in this demo app, the
numbers are always equal.','GetThemedPanelFEQ',ICON:Exclamation)
```

See also:
ThemeAPanel 410
XPThemesPresent 57

3.38.4.32 GetWindowVersion — Windows Class - Methods

Prototype: **(),String,PROC**

Returns Returns windows version in x.x format

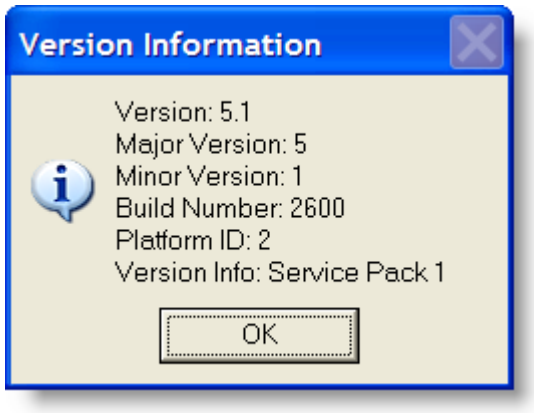
This method uses the GetVersionEx api function to retrieve the windows version information. It returns the Major and Minor version numbers in a x.x format. The method also fills in the MajorVersion, MinorVersion, VersionBuildNr, VersionPlatformID and VersionInformation properties of the ITWindowClass. After a call to this method you can use these properties to get extensive version information about the operating system that your code is running on.

For more information about the windows versions, please see http://msdn.microsoft.com/en-us/library/ms724451(VS.85).aspx and http://msdn.microsoft.com/en-us/library/ms724833(VS.85).aspx

Example:

```
ITW  ITWindowsClass
VS   CString(101)
 Code
 !...
 VS = ITW.GetWindowVersion()
 Message('Version:           ' & VS &|
        '|Major Version:     ' & ITW.MajorVersion &|
        '|Minor Version:     ' & ITW.MinorVersion &|
        '|Build Number:      ' & ITW.VersionBuildNr &|
        '|Platform ID:       ' & ITW.VersionPlatformID &|
        '|Version Info:      ' & ITW.VersionInformation, |
        'Version Information',ICON:Information)
```

On Windows XP Home, service pack 1, the results can be seen in the screenshot below.



See also:
MajorVersion 369
MinorVersion 370
VersionPlatformID 374
VersionBuildNr 373
VersionInformation 374
IsVista 369

3.38.4.33 Is64bitOS

Windows Class - Methods

Prototype: **(),Byte**

Returns True or false depending on if the operating system (OS) is 64bit or not

This method checks if the IsWow64Process function exists in Kernel32.dll. If it does, the operating system can be 64bit and then the function is called and depending on what it returns it is determined if the program is running under 64bits or not.

Example:

```
ITW  ITWindowsClass
 Code

 If ITW.Is64bitOSAvailable()
   Message('This Operating System can be 64bit')
   If ITW.Is64bitOS()
     Message('This program is running under a 64 bit Operating System')
   Else
     Message('This program is running under a 32 bit Operating System')
   End
 Else
   Message('This Operating System cannot be 64bit')
 End
```

See also:
Is64bitOSAvailable 398

3.38.4.34 Is64bitOSAvailable

Windows Class - Methods

Prototype: **(),Byte**

Returns True or false depending on if the operating system (OS) has a 64bit option or not.

This method checks if the IsWow64Process function exists in Kernel32.dll. If it does, the operating system can be 64bit. This includes Windows XP, Windows Vista, Windows Server 2008 and Windows 7, but not older windows versions such as Window 98, Windows 95 or Windows 3.x.

Example:

```
ITW  ITWindowsClass
 Code

 If ITW.Is64bitOSAvailable()
   Message('This Operating System can be 64bit')
   If ITW.Is64bitOS()
     Message('This program is running under a 64 bit Operating System')
   Else
     Message('This program is running under a 32 bit Operating System')
   End
 Else
   Message('This Operating System cannot be 64bit')
 End
```

See also:
Is64bitOS 397

3.38.4.35 IsTerminalServer

Windows Class - Methods

Prototype: **(),BYTE**

Returns Returns true if the session is running on Terminal Server, false if it is not.

This method uses GetSysMetrics 394 to retrieve the setting of the SM_REMOTESESSION, which detects if the session is remote or not.

Example:

```
ITW  ITWindowClass
 Code
 Message('This session is ' & Choose(ITW.IsTerminalServer()=True,'','not')
& ' running on Terminal Server')
```

See also:
GetSysMetrics 394

3.38.4.36 IsProgramRunning

Windows Class - Methods

Prototype: **(String pSemaphoreValue),Byte**

pSemephoreValue Unique string that identifies the program. You could use a GUID or any string

	that you think is completely unique for your program such as 'Icetips ProductName executable'
Returns	True or false depending on if the program is running or not.

This method is used to determine if a program is already running and then activate the currently running instance of it. The method uses CreateSemaphore to create a unique handle to the program instance. If the program is run again, the CreateSemaphore will report that the handle already exists.

This method should be called right after the program CODE statement to prevent any start up code to run. The handle returned by CreateSemaphore is automatically destroyed when the program terminates.

Example:

```
Glo:ITW  ITWindowClass
 Code
 If Glo:ITW.IsProgramRunning('Icetips product')
   Glo:ITW.ActivateWindow('Icetips product appframe')
   Halt()
 End

Glo:ITW  ITWindowClass
 Code
 If Glo:ITW.IsProgramRunning('Icetips product')
   Message('Program is already running, cannot run another instance',
'Program already running')
   Glo:ITW.ActivateWindow('Icetips product appframe')  !! Call AFTER the
Message()
   Halt()
 End
```

See also:
ActivateWindow 378
Limit Program Instance 454
http://msdn.microsoft.com/en-us/library/windows/desktop/ms682438(v=vs.85).aspx

3.38.4.37 MakeLangID — Windows Class - Methods

Prototype:	**(UShort usPrimaryLanguage, UShort usSubLanguage),UShort**
usPrimaryLanguage	Primary Language
usSubLanguage	Sub Language
Returns	Language ID for use with Locale api calls

This method performs the same task as C MakeLangID macro. The value returned can be used in various Locale api functions to retrieve specific language information. For more information on values for Primary Language and Sub Language please check this page on the MSDN website. For more information on national language support api functions check out this page.

Example:

```
(none)
```

See also:
MakeLangID

3.38.4.38 PlaceControlForDPI

Prototype: **(Long pFEQ, <Long pDesignDPI>)**

pPFEQ	Control Field EQuate label
pDesignDPI	Dots Per Inch resolution of the machine used to design the window/control

This method attempts to resize a control based on the original DPI information (if any) or the DPI ratio of the machine it is running on. For example if a control is designed in 120DPI resolution and is being run on a computer with 96DPI resolution, the control may need to be scaled down. This mostly applies to image controls that can look very bad if they are being displayed in a different resolution than they were designed for.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.PlaceControlForDPI(?Image1,96) !! Window designed in 96 DPI.
 ITW.PlaceControlForDPI(?Image2,120) !! Window designed in 120 DPI.
```

See also:
GetScreenDPI 392
GetScreenDPIRatio 393

3.38.4.39 RedrawClientArea

Prototype: **(none)**

This method uses InvalidateRect to force a redraw of the window client area. Sometimes residual bits may be left on a screen, particularly when dealing with non-native Clarion controls. This method takes care of cleaning and redrawing the screen.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.RedrawClientArea
```

See also:

3.38.4.40 RemoveWindowColor

Windows Class - Methods

Prototype: **(),BYTE**

Returns Returns LEVEL:Benign

This method removes background color drawn on a window with the SetWindowColor 408 method. This method, and SetWindowColor, behave differently on an AppFrame window than they do on a standard window. When you use it on an appframe it can only be done when the window opens. On an appframe this method is called automatically when the window closes and should be treated as a private method. On a normal window you can use this method anywhere to go back to the original color of the window.

If SetWindowColor has not been called, this method does not do anything.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetWindowColor(COLOR:White)
 ITW.RemoveWindowColor
```

See also:
SetWindowColor 408

3.38.4.41 ResizeControlForDPI

Windows Class - Methods

Prototype: **(Long pFEQ, <Long pDesignDPI>)**

pFEQ Field EQuate label of the control to resize

[pDesignDPI] The design DPI of the target.

This method resizes the passed control with the ratio of either the DPI Ratio as calculated by GetScreenDPIRatio 393 or based on the pdesignDPI. This can be used to aid in resizing controls to work with different DPI resolutions.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.ResizeControlForDPI(?List1)
```

See also:
GetScreenDPIRatio 393

3.38.4.42 Revert64bitRedirection

Prototype: **(),Byte,PROC**

Returns TRUE if the method was succesfull in enabling 64bit redirection

This method is used to revert or reverse the effects of Disable64bitRedirection 379 which redirects access to registry and file system to use 64bit locations from a 32bit program. See the TestShellClass procedure in the Utilities demo app.

Example:

```
ITS          ITShellClass
FileName     CSTRING(1025)
ProgramFile  CSTRING(1025)
  CODE
  If FileName
    If ITS.Disable64bitRedirection()
      ProgramFile = ITS.GetAssociatedProg(FileName)
      If Not ITS.Revert64bitRedirection()
        Message('Could not change redirection!')
        Stop
      End
    End
```

See also:
Disable64bitRedirection 379

3.38.4.43 SetControlFonts

Prototype: **(Long pFrom, Long pTo)**

pFrom Field EQuate label of a control to use font information from

pTo Field EQuate label of a control to set font information.

This method simply sets all font information for the pTo control exactly the same as the font information for the pFrom control. This includes font name, size, style, color and character set.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetControlFonts(?Button1, ?Button2)
```

3.38.4.44 SetControlPositions

Prototype: **(Long pFrom, Long pTo)**

pFrom Field EQuate label of a control to use position information from

pTo Field EQuate label of a control to set position.

This method simply sets all position information for the pTo control exactly the same as the position information for the pFrom control. This includes the X and Y coordinates as well as Height and Width. Basically it places one control in exactly the same as the other control and makes them the same size.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetControlPostions(?Button1,?Button2)
```

3.38.4.45 SetControlProp

Windows Class - Methods

Prototype: **(Long pFEQ, Long pProperty, String pValue)**

pFEQ Field EQuate label for the control to change

pProperty Property to set

pValue Value to set the Property to.

This method does simple property assignment, but only if the property value has changed. This can reduce flicker caused by setting properties on controls repeatedly inside of a loop. This method simply sets the property only if it has changed.

Example:

```
ITW  ITWindowsClass
 Code
 If EVENT() = EVENT:Timer
   If TimerActive
     Loop 100 Times
       Next(MyFile)
       If ErrorCode()
         TimerActive = False
         Close(MyFile)
         Break
       End
       ITW.SetControlProp(?String1,PROP:Text,MYF:Name) ! Only changes
PROP:Text if MYF:Name has changed
     End
   End
 End
```

3.38.4.46 SetPixelHeight

Windows Class - Methods

Prototype: **(Long pFEQ, Long pValue)**

pFEQ The Field EQuate label of the control to set height for.

pValue The height of the control to set in pixel

This method uses the SetPixelPos 404 method to set the height of a control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelHeight(?Button1,25)  !! Set the button 25 pixels tall.
```

See also:
GetPixelHeight 388
SetPixelPos 404
SetPixelWidth 405
SetPixelXPos 406
SetPixelYpos 406

3.38.4.47 SetPixelPos

Prototype: **(Long pFEQ, Long pC, Long pValue)**

pFEQ The Field EQuate label of the control to set height for.
pC The property to set. This can be PROP:Height, PROP:Width, PROP:Xpos or PROP:Ypos
pValue The height of the control to set in pixel

This method sets the appropriate property such as height, width, x-position or y-position in pixels. It is used by SetPixelHeight 403, SetPixelWidth 405, SetPixelXPos 406 and SetPixelYPos 406.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelPos(?Button1, PROP:Heigth, 25)  !! Set the button 25 pixels
tall.
```

See also:
SetPixelHeight 403
SetPixelWidth 405
SetPixelXPos 406
SetPixelYpos 406

3.38.4.48 SetPixelPosition

Windows Class - Methods

Prototype: **(Long pFEQ,<Long pX>,<Long pY>,<Long pW>,<Long pH>)**

pFEQ	The Field EQuate label of the control to set position for.
pX	The X coordinates for the control in pixels.
pY	The Y coordinates for the control in pixels.
pW	The Width of the control in pixels.
pH	The Height of the control in pixels.

This method is basically identical to the Clarion SetPosition statement except it uses Pixel positioning rather than Dialog Unit positioning. This allows you to precisely place a control or set the size of a control with one method call.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelPosition(?Button,100,50,75,25)  !! set size of ?Button to 100
pixels from left edge, 50 from top edge, 75 pixels wide and 25 tall.
```

See also:

SetPixelHeight 403

SetPixelWidth 405

SetPixelXPos 406

SetPixelYPos 406

SetPixelPos 404

3.38.4.49 SetPixelWidth

Windows Class - Methods

Prototype: **(Long pFEQ, Long pValue)**

pFEQ	The Field EQuate label of the control to set width for.
pValue	The width of the control to set in pixel

This method uses the SetPixelPos 404 method to set the width of a control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelWidth(?Button1,100)  !! Set the button 100 pixels wide.
```

See also:

GetPixelWidth 390

SetPixelPos 404

SetPixelWidth 405
SetPixelHeight 403
SetPixelXPos 406
SetPixelYpos 406

3.38.4.50 SetPixelXPos

Windows Class - Methods

Prototype: **(Long pFEQ, Long pValue)**

pFEQ	The Field EQuate label of the control to set X-position for.
pValue	The X coordinates of the control to set in pixel

This method uses the SetPixelPos 404 method to set the X-position of a control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelXPos(?Button1,100)  !! Set the button 100 pixels in from the
left window edge.
```

See also:
GetPixelXPos 390
SetPixelPos 404
SetPixelWidth 405
SetPixelHeight 403
SetPixelYpos 406

3.38.4.51 SetPixelYPos

Windows Class - Methods

Prototype: **(Long pFEQ, Long pValue)**

pFEQ	The Field EQuate label of the control to set Y-position for.
pValue	The Y coordinates of the control to set in pixel

This method uses the SetPixelPos 404 method to set the Y-position of a control in pixels.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetPixelYpos(?Button1,50)  !! Set the button 50 pixels down from the
top edge of the window.
```

See also:

GetPixelXPos 390
SetPixelPos 404
SetPixelWidth 405
SetPixelHeight 403
SetPixelXpos 406

3.38.4.52 SetToolboxCaption

Prototype: **(Long pHwnd, Byte pSetOn=True)**

pHwnd Handle of a window to set toolbox caption on.

pSetOn Indicates if the effect is to be turned on or off. True turns it on, False turns it off.

This method sets a toolbox caption for the window. A toolbox caption has a smaller caption bar but also can only have the close button and nothing else, i.e. no minimize, maximize buttons or system menu are visible. This works well for toolbox windows. This method resizes the window to match the correctly small and large caption bars as reported by GetSysMetrics 394 with the SM_CYCAPTION and SM_CYSMCAPTION parameters.

Note:
Even though this method takes the handle of the window as a parameter, it presumes that the window is also the current target, i.e. it uses 0 as window reference to get and set the size of the window. If you are using this in a situation where the window is not the current target, please keep this in mind and use **SetTarget** before and after calling this method.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetToolboxCaption(0{Prop:Handle} , Choose(Loc:IsToolbox=True, False,
True))
 Loc:IsToolbox = Choose(Loc:IsToolbox=True, False, True)
```

This will toggle the window between being a toolbox window and being a normal window. All styles of the window are preserved so that when the toolbox is turned off, all properties of the window as it was originally are restored.

Normal window:

Same window with SetToolboxCaption turned on:

See also:
GetSysMetrics 394

3.38.4.53 SetWindowColor

Prototype: **(Long pColor)**

pColor Color to use

When this method is used on an appframe window it should be called during the startup process of the window i.e. in ThisWindow.Init, after the window is opened (priority around 8000).

When this method is used on other windows it can be called anywhere and at any time after the window is opened and called repeatedly.

Example:

```
ITW  ITWindowsClass
Col  Long
 Code
 If ColorDialog('Select Color',Col)
   ITW.SetWindowColor(Col)
 End
```

See also:
RemoveWindowColor 401

3.38.4.54 SetWindowNotOnTop

Prototype: **None**

This method can be used to make the window not be on top after it is set with SetWindowOnTop.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetWindowNotOnTop
```

See also:
SetWindowOnTop 408

3.38.4.55 SetWindowOnTop

Prototype: **None**

This method can be used to set windows to be on top of other windows.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetWindowOnTop
```

See also:
SetWindowNotOnTop 408

3.38.4.56 SetWindowPosition — Windows Class - Methods

Prototype: **(Long pX, Long pY, Byte pSetPixels=True)**

pX The X coordinate for the window

pY The Y coordinate for the window

pSetPixels Indicates if PROP:Pixels is turned on during the positioning

This method does the same thing as the clarion Position() function, but can alternatively be set to use pixels for more accurate placement.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetWindowPosition(0,0,True)
```

See also:
SetWindowSize 409

3.38.4.57 SetWindowSize — Windows Class - Methods

Prototype: **(Long pWidth, Long pHeight, Byte pSetPixels=True)**

pWidth The width of the window

pHeight The height of the window

pSetPixels Indicates if PROP:Pixels is turned on during the sizing

This method does the same thing as the clarion Position() function, but can alternatively be set to use pixels for more accurate sizing.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.SetWindowSize(800,600,True)
```

See also:
SetWindowPosition 409

3.38.4.58 ThemeAPanel

Prototype: **(Long pPanelFEQ)**

pPanelFEQ The Field EQuate label of the panel to theme

This method themes a panel using the XP Theme methods.

NOTE: This is only going to work if your application is using the XP Theme product from www.cwtemplates.com

Example:

```
ITW  ITWindowsClass
 Code
 ITW.ThemeAPanel(?Panel1)
```

3.38.4.59 UsesClearType

Prototype: **(),Byte**

Returns Returns True if the system is using Clear Type

This method checks a registry key and returns true if it is set to use Clear Type font smoothing. This can come in handy for Clarion 6 and older which do not correctly support Clear Type fonts in entry fields. This problem is fixed in Clarion 7. This code was inspired by information found here.

Example:

```
I Long
ITW  ITWindowsClass
 Code
 If ITW.UsesClearType
   Loop I = FirstField() To LastField()
     If I{Prop:Type} = CREATE:ENTRY
       I{Prop:FontName} = 'MS Sans Serif'
       I{Prop:FontSize} = 8
     End
   End
 End
```

3.38.4.60 UsingLargeFonts

Prototype: **(),Byte**

Returns Returns True if the system is using larger than 100% fonts, false if it is not.

This method uses a very simply formula to detect if the fonts used are larger than 100% i.e. using large fonts. It checks if the horizontal dialog units are more than 16 pixels wide. If so the return value is true otherwise it is false.

Example:

```
ITW  ITWindowsClass
 Code
 If ITW.UsingLargeFonts()
   ?Button{Prop:FontName} = 'Microsoft Sans Serif'
   ?Button{Prop:FontSize} = 8
 Else
   ?Button{Prop:FontName} = 'Microsoft Sans Serif'
   ?Button{Prop:FontSize} = 9
 End
```

See also:
GetDialogUnit 385

3.38.4.61 WindowInfoToODS

Windows Class - Methods

Prototype: **(String pProcedureName),VIRTUAL**

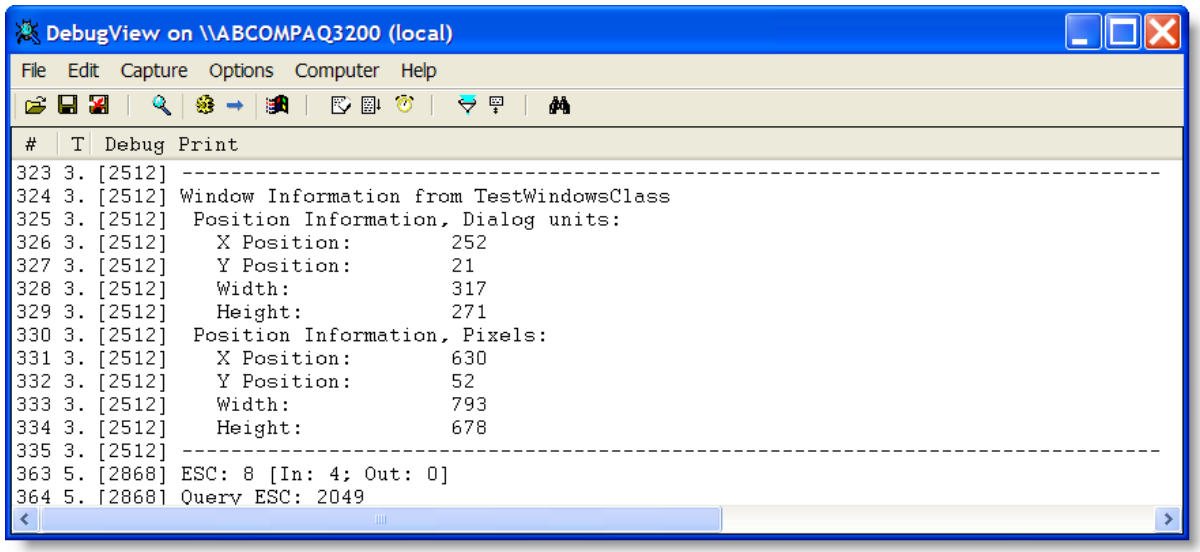
pProcedureName Name of the procedure where the window is.

This method sends position information in both dialog units and pixels to OutputDebugString using the ODS 75 method.

Example:

```
ITW  ITWindowsClass
 Code
 ITW.WindowInfoToODS('MyProcedure')
```

Results in this capture in DebugView:



See also:
ODS 75

3.38.4.62 Construct — Windows Class - Methods

Prototype: **None**

This constructor creates new instances of the ChildWindowQ 367 as ChildWindows 368 and TopWindows 372 properties. It also creates a new

```
 SELF.ChildWindows  &= NEW ChildWindowQ
 SELF.TopWindows    &= NEW ChildWindowQ
 SELF.ModuleWindows &= NEW ChildWindowQ
 IT_PROCESS_ALL_ACCESS = IT_STANDARD_RIGHTS_REQUIRED + IT_SYNCHRONIZE +
IT_PROCESS_ALL_ACCESS_ADDIN
 V = SELF.GetWindowVersion()
 SELF.ThemedControls &= NEW (tThemedControls)
```

See also:
Destruct 412

3.38.4.63 Destruct — Windows Class - Methods

Prototype: **None**

This Destructor cleans up the ChildWindows 368 and TopWindows 372 property queues.

See also:
Construct 412

3.38.5 Procedures — Windows Class

The Windows class contains one enumeration procedure that is a callback procedure for the EnumChildWindows API call used to enumerate the child windows in EnumChildWin 380. It also contains a copy of that procedure that is a callback procedure to enumerate top windows.

3.38.5.1 EnumTopWindowsProc — Windows Class - Procedures

Prototype:	**(Long hwnd, Long lParam),BOOL,PASCAL**
hwnd	Handle to the window
lParam	Reference to the Windows class
Returns	True

This is a standard enumeration procedure for the EnumWindow API call defined as EnumWindowsProc *(links active as of June 13, 2007)*

See also:

EnumChildWin 380

3.38.5.2 EnumChildWindowsProc

Windows Class - Procedures

Prototype:	**(Long hwnd, Long lParam),BOOL,PASCAL**
hwnd	Handle to the window
lParam	Reference to the Windows class
Returns	True

This is a standard enumeration procedure for the EnumChildWindow API call defined as EnumChildProc *(links active as of June 13, 2007)*

See also:

EnumChildWin 380

Part IV

Chapter 4 - Templates

4 Templates

As of May 14, 2016, the following templates are defined in the Icetips Utilities:

Code templates

ITAssignCSIDL 418	Icetips Utilities: Assign Special Folder CSIDL
ITCreateFileViewCode 418	Icetips Utilities: Create File View Code
ProceduresInQueue 417	Icetips Utilities: Add Procedures To Queue
StoreClarionBuildInVariable 421	Icetips Utilities: Store Clarion Build in a variable
StoreCompileDateInVariable 422	Icetips Utilities: Store compile date/time in variables

Control templates

ITUPageOfPagesControl	Icetips Utilities: Page of Pages
IcetipsMSWindowHeader	Icetips Utilities: MS Window header

Global Extension templates

AddCompileDateTimeToVersion 430	Icetips Utilities: Add Compile Date/Time to version
AddVistaManifest 432	Icetips Utilities: Add Windows Manifest to application
GlobalAlertOnLookups 437	Icetips Utilities: Global Alert on Lookup controls
GlobalCallShowRecordToBrowse 439	Icetips Utilities: Global Call ShowFileRecord from Browse
GloballyCallProcedure 435	Icetips Utilities: Call procedure from all procedures
~~GloballyFixWindowXYPos~~	~~Fix Window X/Y Positions - DEPRECATED~~ This template is no longer used. Use Window Fixer instead
ITDLLLoaderDLL	Icetips Utilities: DLL Loader - DLL Application
ITExportTXATXD 442	Icetips Utilities: Export App and Dct
ITGlobalAliasFiles 448	Icetips Utilities: Alias Files
ITGlobalBreadCrumbs	Icetips Utilities: Breadcrumbs Global
ITGlobalGenerateFileQueue 443	Icetips Utilities: Generate File Queue
ITGlobalHideWindowWhileLoading 450	Icetips Utilities: Hide Windows while loading
ITGlobalLimitProgramInstances 454	Icetips Utilities: Limit Program Instance
ITGlobalThreadedWindowManager 449	Icetips Utilities: Global Threaded Window Manager
ITIncludeExpFiles 453	Icetips Utilities: Include Export (.exp) files
ITUtilitiesGlobal 451	Icetips Utilities Classes Global - ABC ONLY
ITUtilitiesGlobalLegacy 451	Icetips Utilities Classes Global - LEGACY ONLY
WriteTemplateInfoToFile 456	Icetips Utilities: Write Template info to file
WriteVersionInfoToINIFile 456	Icetips Utilities: Write Version info to INI File

Procedure Extension templates

ITAddHeaderSortTo Queue 458	Icetips Utilities: Add Header Sort to Queue
ITBindUnbindLocalVar 460	Icetips Utilities: Bind/Unbind local variables
ITBrowseUpdateChe	Icetips Utilities: Browse Checkbox update

ckbox 465	
ITCreateFileView	Icetips Utilities: Create File View
ITDLLLoaderEXE	Icetips Utilities: DLL Loader - EXE Main Procedure
ITDuplicateWindow 463	Icetips: MDI/SDI Duplicate Window Structure
~~ITFillQueueFromView~~	~~Icetips Utilities: Fill Queue from SQL View - DEPRECATED~~
ITLoadQueueFromView	Icetips Utilities: Load Queue from View
ITLocalProcedures	Icetips Utilities: Local Procedures
ITPrePostPrimeABC 470	Icetips Utilities: Pre and post prime ABC Browse
ITPreserveVariableData 474	Icetips Utilities: Preserve Variable Data
ITProcThreadLimiter 471	Icetips Utilities: Thread Limiter - Procedure
ITProcThreadedWindowManager 466	Icetips Utilities: Call Threaded Window Manager
ITResizeOptions 470	Icetips Utilities: Resize Options
ITResizeOptionsWithInfo 470	Icetips Utilities: Resize Options With Information
ITSQLQueueProcess 471	Icetips Utilities: SQL Queue Process Construction
ITSQLQueueReport 471	Icetips Utilities: SQL Queue Report Construction
~~ITSQLReportQueue~~	~~***DO NOT USE***: Icetips SQL Report Queue Construction~~
ITWriteProcedureInfoToFile	Icetips Utilities: Write Procedure information to File

Utility templates

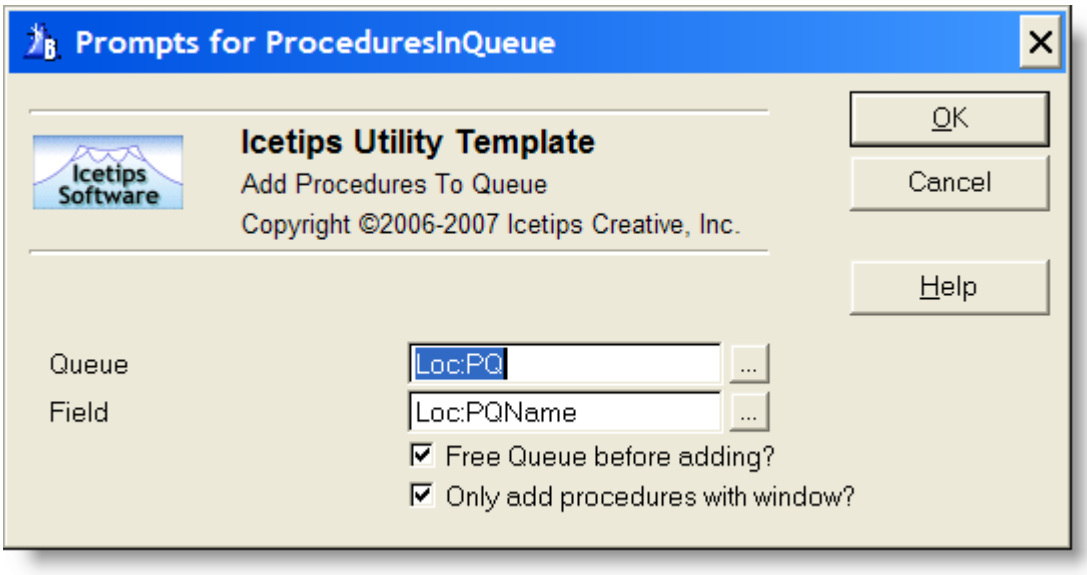
IcetipsWinCodeWizard 491	Icetips Utilities: Standardized Window Code Wizard
ITExportWindowsWithoutHLP 484	Icetips Utilities: List procedures with Windows without HLP attribute
ITPrepareMultiDLL 496	Icetips Utilities: Prepare Multi-DLL app
ITShowFileRecordWizard 488	Icetips Utilities: ShowFileRecord Wizard
ITWindowWizard 479	Icetips Utilities: Create a New Window Procedure
ITWriteFilesToFile 498	Icetips Utilities: Write Used/Unused Files to file
ITWriteTemplatesToFile 498	Icetips Utilities: Write Templates to file
ITWriteTemplatesToFileCompact 498	Icetips Utilities: Write Templates to file (compact form)
ITWriteIconAndImagesToFile 505	Icetips Utilities: Write Icon and Image information to file
WriteProcedureModulesToFile 509	Icetips Utilities: Write Modules and procedure information to File

4.1 Code Templates

The Icetips Utilities currently contain 5 code templates:

4.1.1 Add Procedures To Queue — Code Templates

This code template loads the names of all procedures in the application into a local queue.



This template allows you to specify which queue and field to fill with information.

Queue — The label of a queue which will be loaded with the procedure names.

Field — The label of a field in the Queue which will receive the procedure name.

Free Queue... — When this is checked, a FREE() is executed on the queue just before it is filled. If this is not checked, the queue is not FREEd. This is checked by default.

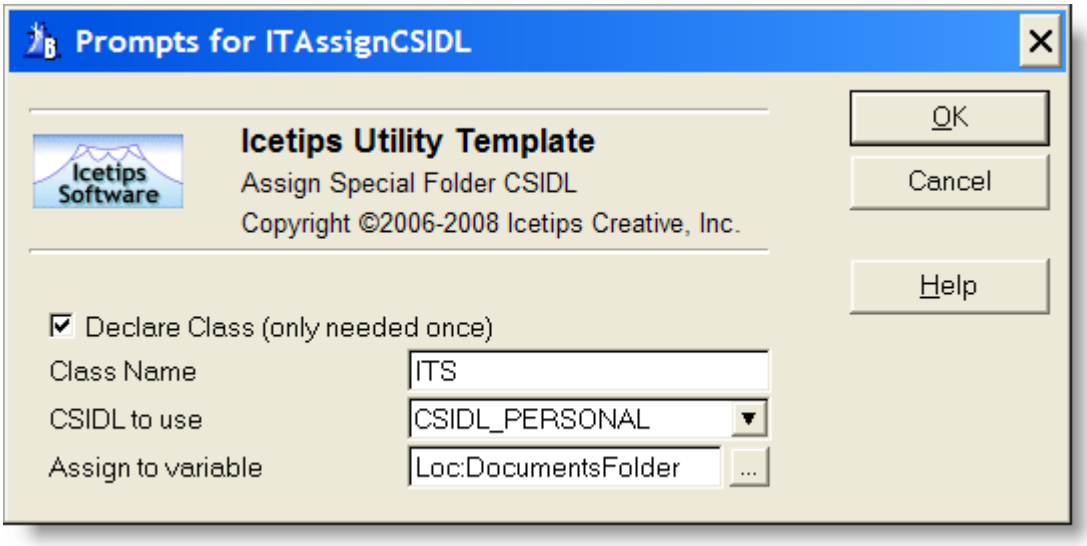
Only add procedures... — When this is checked the template will add procedures with a window ONLY. This will exclude source procedures and any other procedures that do not have a window. Note that report procedures will be included if they also have a progress window. This is checked by default.

The use of this template is demonstrated in the TestTemplate 515 procedure in the UtilDemo.app 514.

See also:
Example app: TestTemplate 515

4.1.2 Assign Special Folder CSIDL Code Templates

This code template declares a ShellClass 257 instance and assigns the value of a CSIDL folder to a selected variable



This template allows you to optionally declare the ShellClass 257 in your procedure. You only need to do that once per procedure.

Declare Class Determine if the class should be declared or not. If this is unchecked the class is not declared.

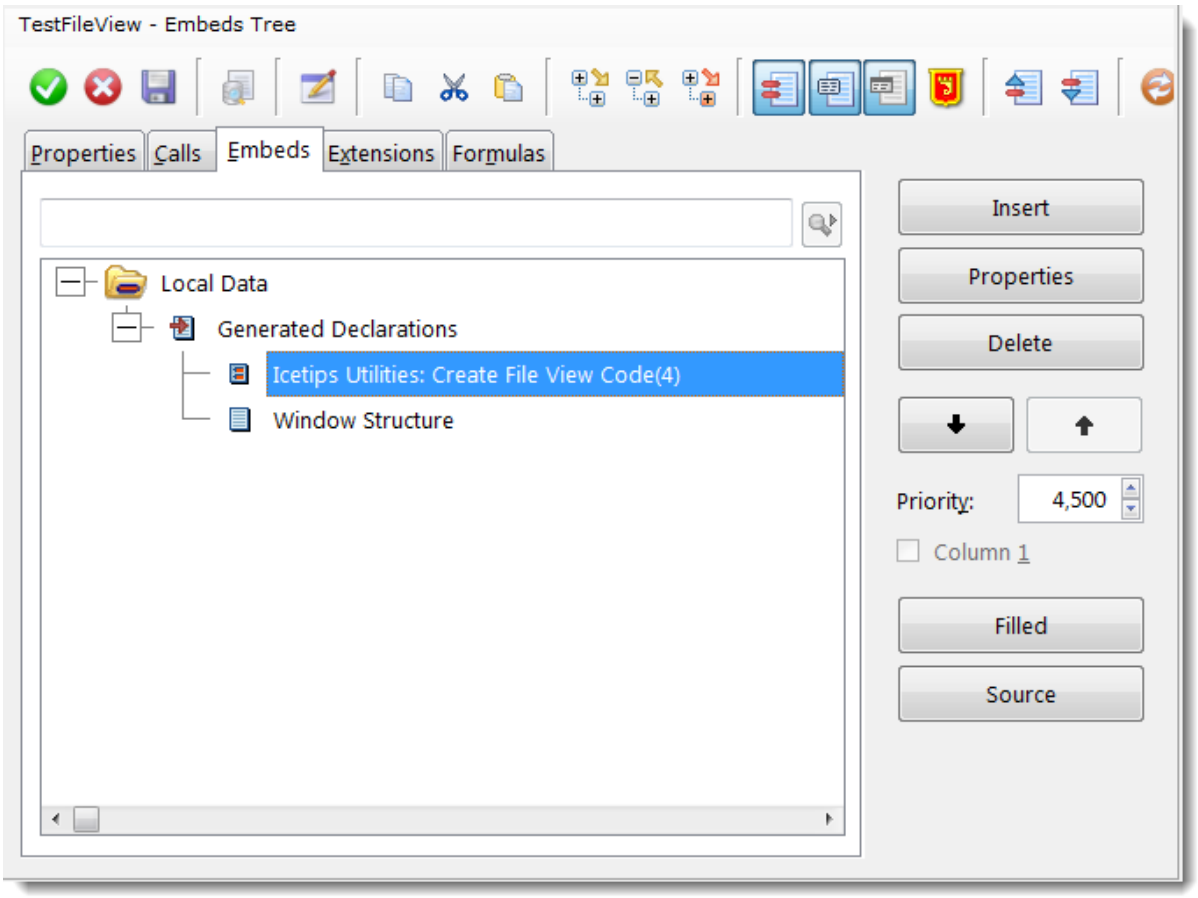
Class Name The name of the class to declare or use. If the "Declare Class" is checked this is the label of the class that will be declared. If "Declare Class" is not checked, this is the label of the class that will be used. This entry is required.

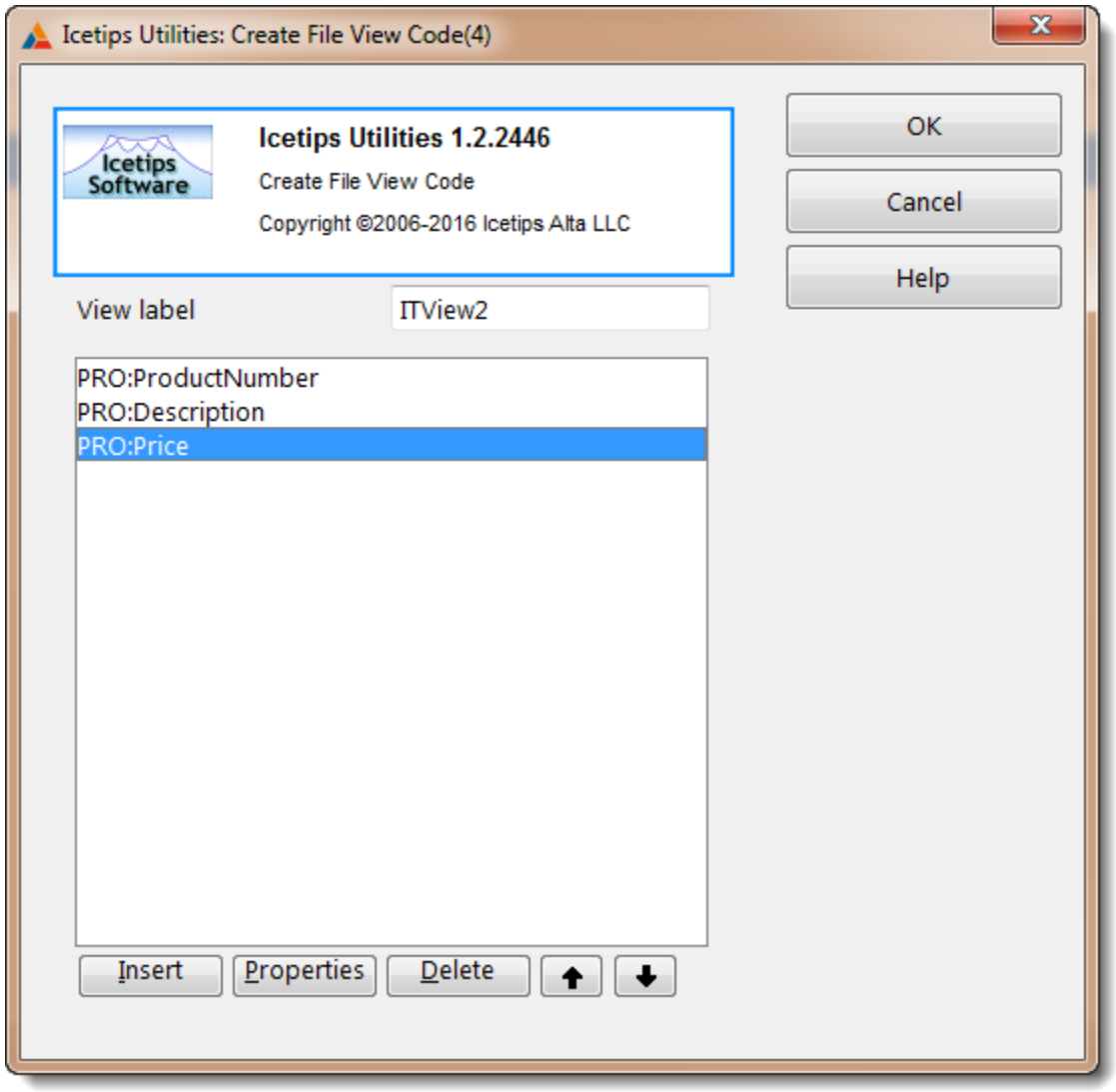
CSIDL to use Dropdown of CSIDL available in the ITWin32Equates.inc file for CSIDLs. The list is in alphabetical order.

Assign to variable Select a variable to assign the value to. This is required.

4.1.3 Create File View Code Code Templates

This template constructs a Clarion VIEW structure in the selected embed. Normally you would use this in one of the "Local Data" embed points as shown in the screenshot below:

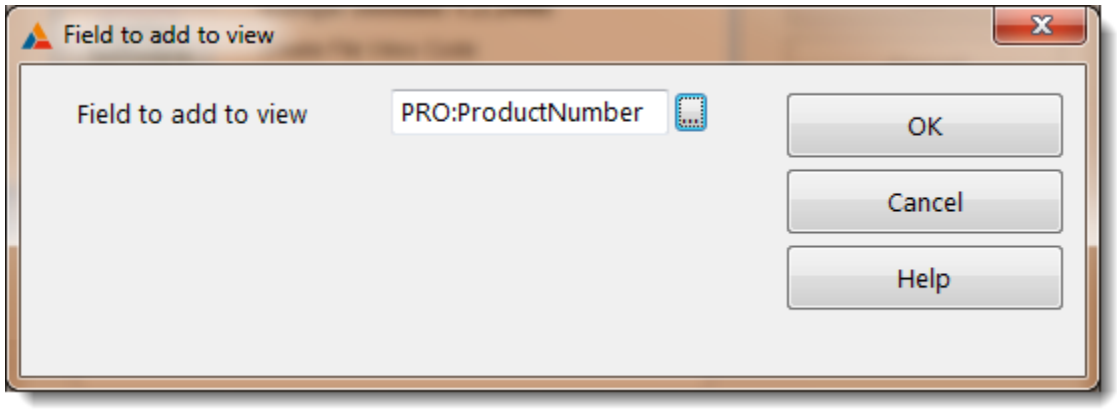




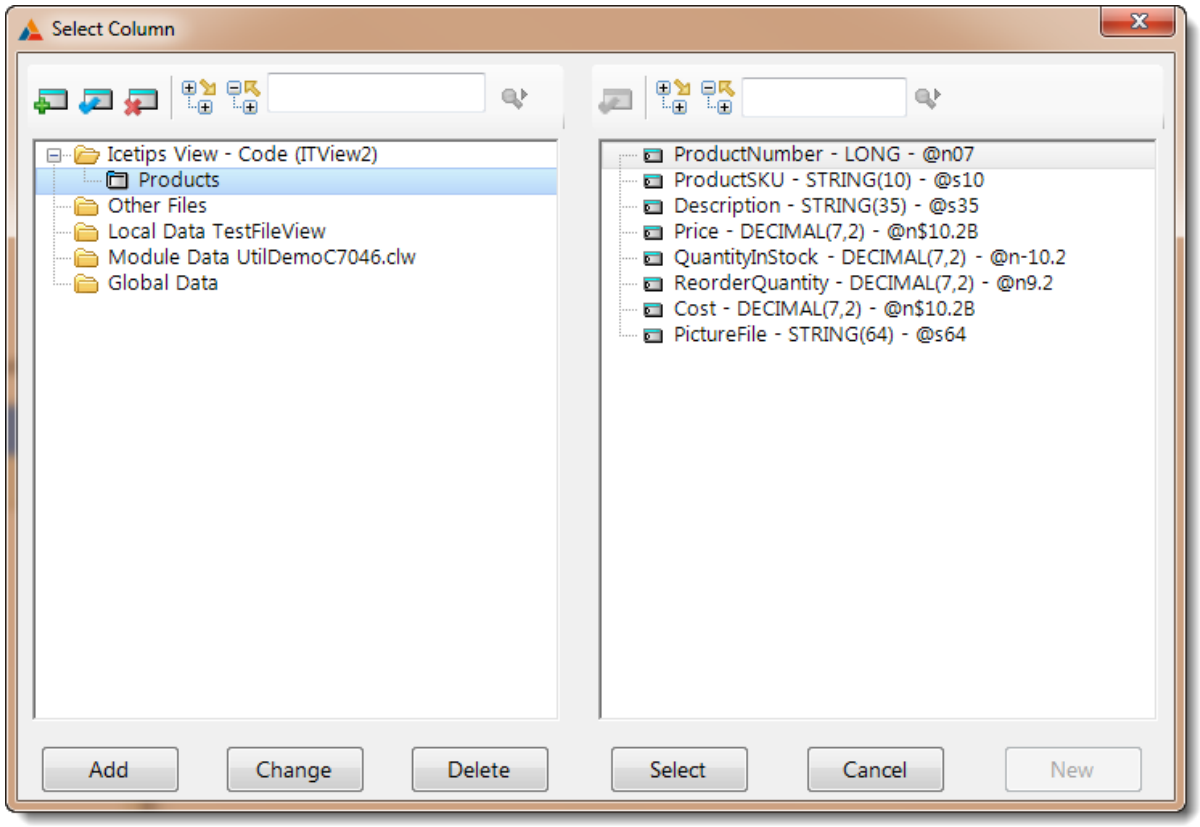
View label This is the label of the view as it is created in the embed.

Field list This is a list of the fields being added to the view. The list above results in a view declared as follows:

```
ITView2                VIEW(Products)
                         PROJECT(PRO:ProductNumber)
                         PROJECT(PRO:Description)
                         PROJECT(PRO:Price)
                       END
```



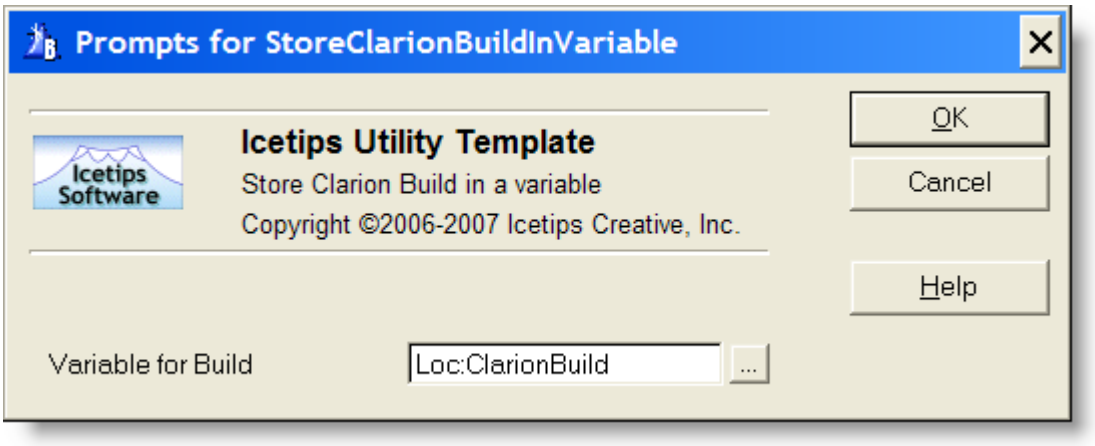
Field to add to view Select the field to add by clicking on the [...] button on the right. This will bring up the default "Select Column" window where you can select the field.



See also:
Create File View extension 467

4.1.4 Store Clarion Build in a variable — Code Templates

This template stores the value of the %CWVersion template symbol in a variable.

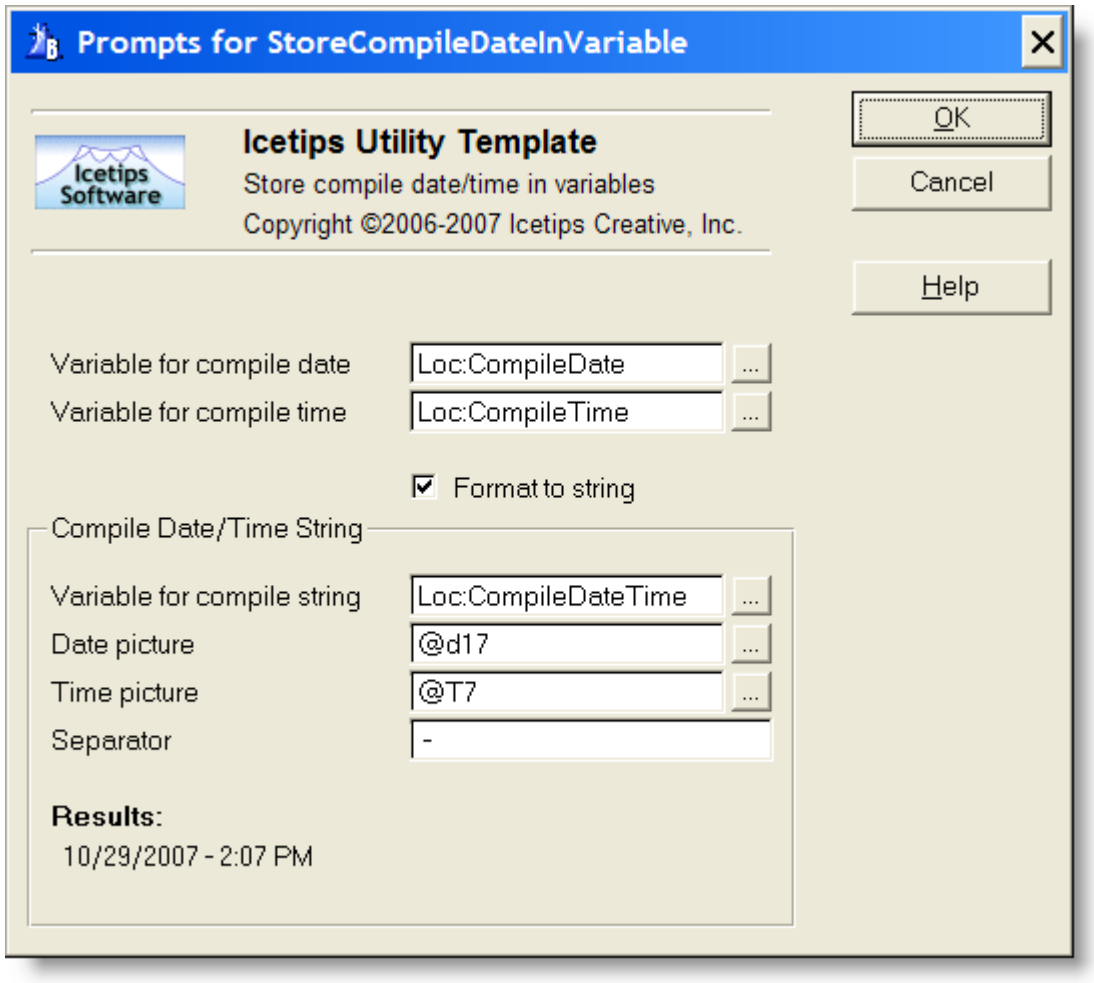


Variable for Build This variable receives the build number. The build number is a 4 digit integer that ranges from 2000 for Clarion for Windows 2.0 to 8000 for Clarion 8. Note that the build number that Softvelocity issues for each of their builds is not included in this version information. I.e. Clarion 6.3 build 9053 shows as 6300. Same does Clarion 6.3 build 9057.

4.1.5 Store compile date/time in variables Code Templates

This template will store the compile date and/or time in variables that you specify. It can also store the concatenated date/time in a single string variable formatted the way you want it.

Note that this may not get updated if the procedure you add this to hasn't changed. One way to force the update would be to delete the source module for this procedure before you generate and compile the application. If you are using automated building process, such as Build Automator it makes it easier to do that.



Variable for compile date This variable receives the compile date value. This value is created when the application code is generated and does not change at runtime. This variable should be a DATE or a LONG variable.

Variable for compile time This variable receives the compile time value. This value is created when the application code is generated and does not change at runtime. This variable should be a TIME or a LONG variable.

Format to string When this checkbox is checked, it enables the "Compile Date/Time String" group below. This allows you to place the formatted compile date and or time into a single string variable to place on a window or a report.

Variable for compile string This variable receives the formatted compile date and/or time values depending on the pictures specified for each variable. This should be a STRING or CSTRING variable. The required size varies based on format picture that you choose.

Date picture The format picture for the date. Type in a different picture if you need or use the [...] button to select a new picture.

Time picture The format picture for the time. Type in a different picture if you need or use the [...] button to select a new picture.

Separator String that is placed between the date and the time.

Results Shows the formatted date and time as it will appear on the window or report.

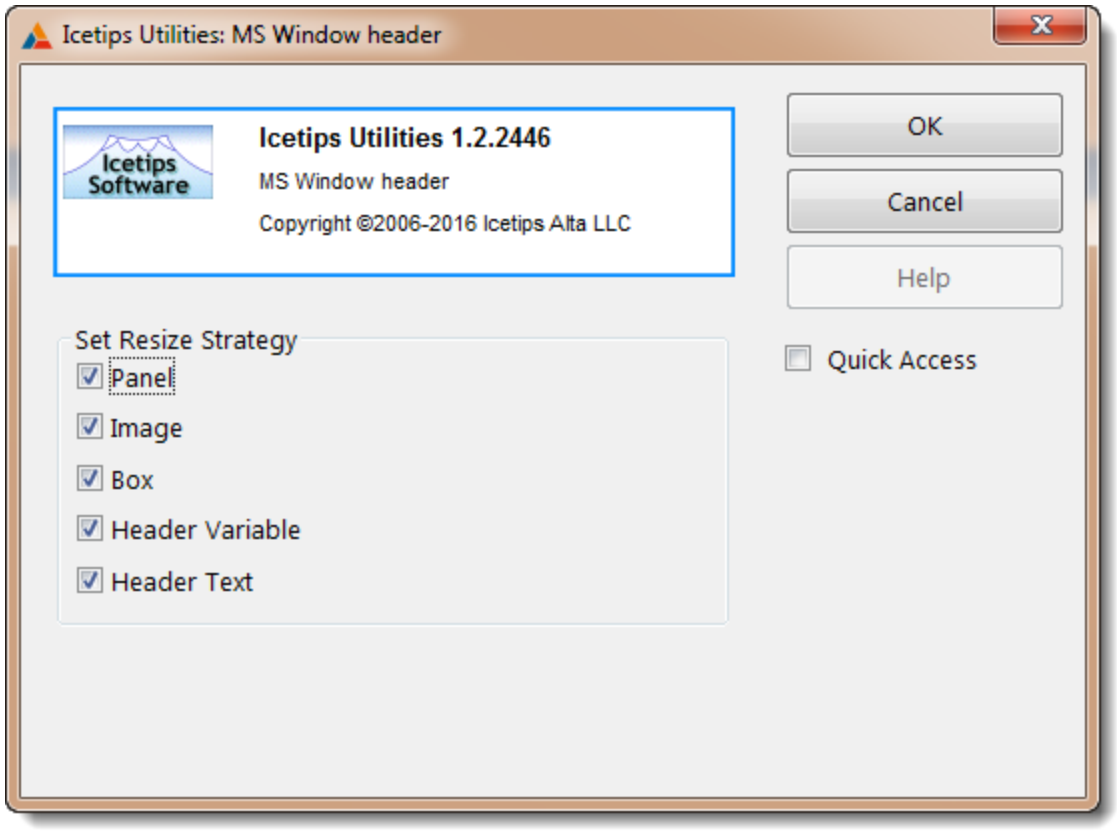
See also:
Example app: TestTemplate 515

4.2 Control Templates

The Icetips Utilities currently contain 2 control templates:

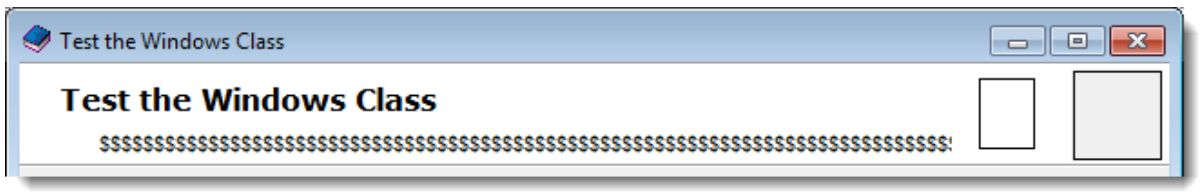
4.2.1 Icetips MS Window header — Control Templates

This control template adds 4 controls at the top of the window, two string controls, one image control and one box control. It lets you create a nice header on your window with some additional information and an icon/image as well.

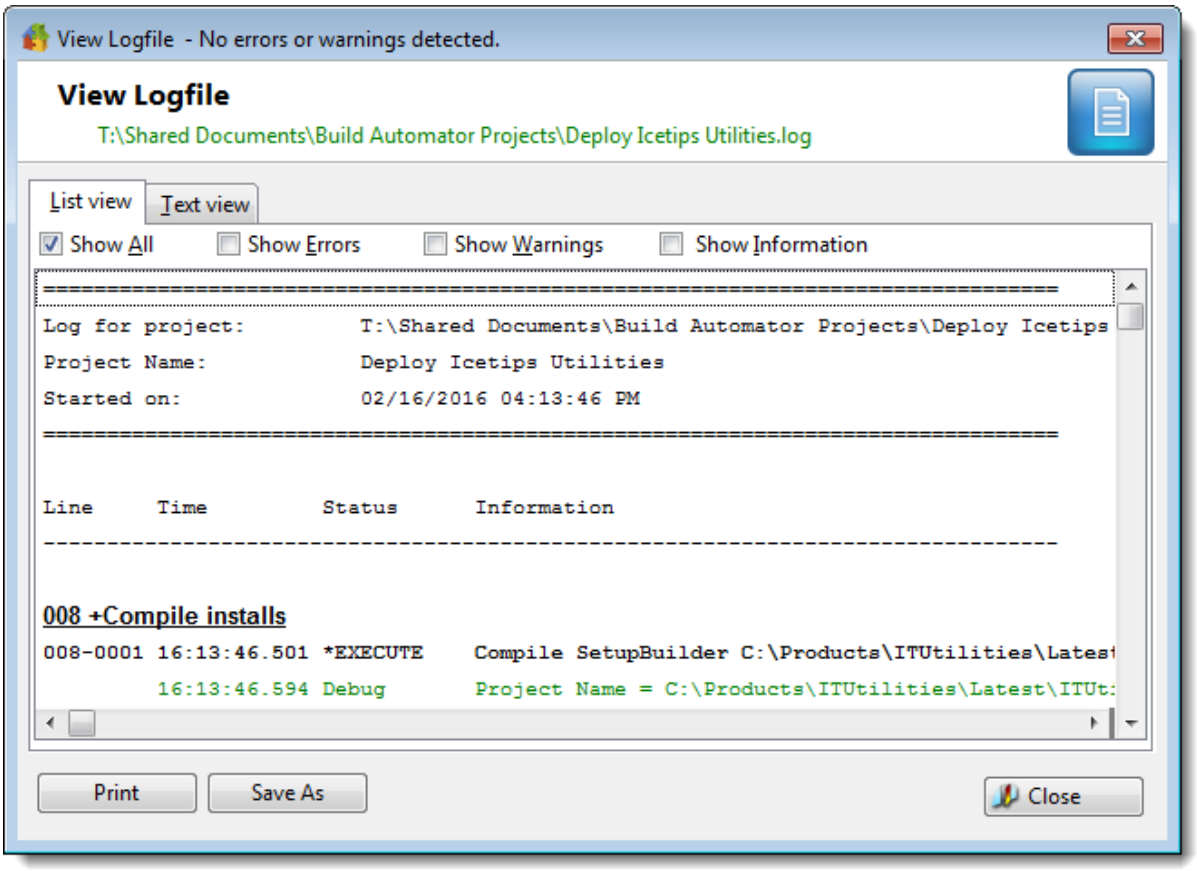


Set Resize Strategy — This lets you specify if you want to set the resize strategy for the controls. Normally all should be checked, but if you don't want some of them to be resized, then uncheck them and they will stay put.

This is how it looks in the window designer.



And this is an example of how it looks like at runtime (screen shot from Build Automator Logfile window)

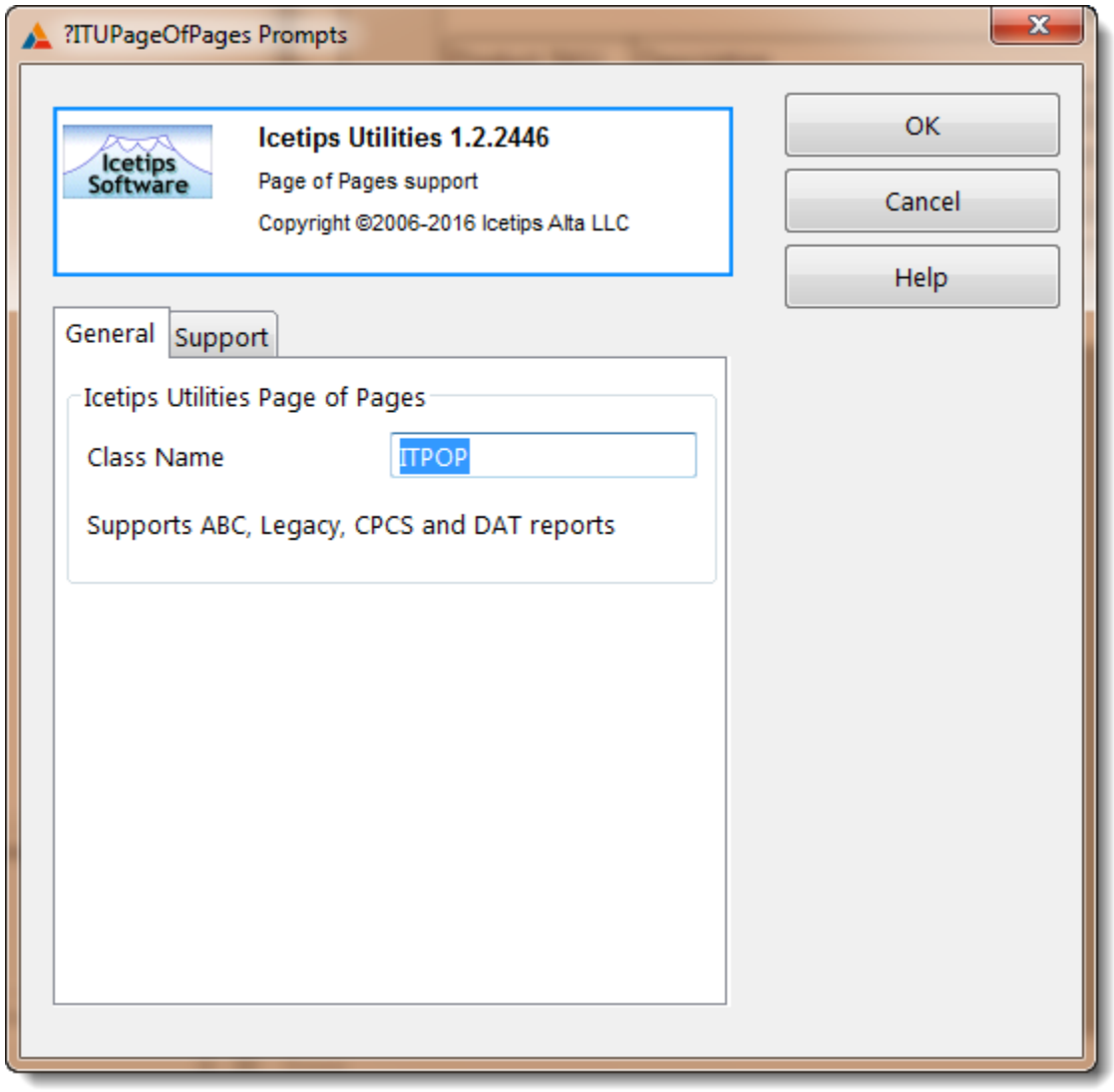


For an example, see the TestWindowsClass procedure in UtilDemo.app and UtilDemoC7.app.

4.2.2 Page Of Pages Template Control Templates

The Page of Pages class can be used with any kind of Clarion report. It can be implemented in hand coded reports as well as in reports in applications. Both Icetips Previewer and Icetips Utilities includes a template that makes it easy to implement on reports.

As of June 2012, the Icetips Utilities also include a template to implement the Page of Pages class on your reports. However, if you are using the Icetips Previewer we suggest that you use that template as it is specifically designed for the Previewer.



Class Name Name of the ITPageOfPages class instance to use in the procedure.

This control template creates a string control that by default uses "?PPPP?" as the text of the control.

The "Page <<<# of" control is a page number control defined with a picture of "@pPage <<<# of p" and shows up in the report structure as:

```
STRING(@pPage <<<# of p),AT(5625,8,865,208),PAGENO,USE(?PageCount),FONT('Arial',10,,),
```

The ?PPPP? control immediately to the right of the page number control is a normal string control that is used by the Page of Pages class. This control will get the total number of pages to place into the control. If it is dropped on by the Previewer control template, then it will be created like this in the report structure:

```
STRING('?PPPP?'),AT(6500,17),USE(?ITPPageOfPages),LEFT,#SEQ(2),#ORIG(?ITPPageOfPages)
```

It it was dropped by this template the only difference is in the USE, which will be ?ITUPageOfPages instead of ?ITPPageOfPages

For a normal report where you want the Page Of to be the full range of the report, that is all you need to do!

However, if you want to control how many pages are in certain ranges, then all you need to do is call the ITPOP.SetPageOfPages at that point. For example:

```
If PRO:ProductSKU[1] > 'F'
   If PRO:ProductSKU[1] <> Clip(Loc:LastProduct)
     Print(RPT:PageBreakDetail)
     ITPOP.SetPageOfPages
     Loc:LastProduct = PRO:ProductSKU[1]
   End
 End
```

If you want to add this to a hand coded procedure or if you don't want to use the template, it is also very easy. On the report procedure you need to add a few lines of code.

1. In the Local data embed add:

```
ITPOP  ITPageOfPagesClass
```

2. In the ThisWindow.OpenReport embed, after Parent Call, put:

```
  ITPOP.Init(Report,SELF.PreviewQueue,'?PPPP?')
```

The first parameter is the report structure label, then it is the image queue used for the metafile pages. The '?PPPP?' string in the final parameter MUST match the string on the report, i.e. the STRING('?PPPP?')

3. Immediately before the report is previewed, such as the ThisWindow.AskPreview embed, add:

```
  ITPOP.SetPageOfPages
```

That's it! If you are using the Icetips Previewer or Icetips Utilities control template, all you need to do is drop the control template on the report and you are done.

If you want to reset the total page number, for example if you are printing invoices that can multiple pages, you can simply call the SetPageOfPages 198 method after printing each detail. The class keeps track of which imagefiles it has updated and will not check any files that have already been updated. The SetPageOfPages is very fast and you will not see much impact from this on your report performance. Here is an example of code put into ThisWindow.TakeRecord after Parent call, to split a report up based on the first letter in the product name, if the name starts with the letter 'G' or later:

```
If PRO:ProductSKU[1] > 'F'
   If PRO:ProductSKU[1] <> Clip(Loc:LastProduct)
     Print(RPT:PageBreakDetail)
     ITPOP.SetPageOfPages
     Loc:LastProduct = PRO:ProductSKU[1]
```

```
   End
 End
```

For a pure hand coded report, something like this should work:

```
PQ   Queue(PreviewQueue)
     END
R    REPORT,AT(1000,1000),PREVIEW(PQ),PAPER(PAPER:LETTER),THOUS,PRE(RPT)
         HEADER,AT(1000,1000,7500,350),USE(?HEADER1)
         END
detail1  DETAIL,AT(0,0,8500,217),USE(?DETAIL1)
         END
         FOOTER,AT(1000,9000,7500,275),USE(?FOOTER1)
             STRING('?PPPP?'),AT(6217,25),USE(?STRING1)
         END
     END
ITPOP  ITPageOfPagesClass
  CODE
  Open(Report)
  ITPOP.Init(Report,PQ,'?PPPP?')
  SET(SomeFile)
  Loop
    Next(SomeFile)
    If ErrorCode()
      BREAK
    END
    PRINT(RPT:Detail)
  End
  ITPOP.SetPageOfPages
  Close(Report)
```

Is it possible that this will fail if the metafile happens to have the same string in it as you use in your INIT method? Yes, it is possible. However with a string like ?PPPP? it is very unlikely that you will ever run into problems with it. To prevent possible problems with changing the token, it is only replaced if it is only found once and only once in the metafile. So if the search string combination is found twice, then the search token will not be update with the total page count.

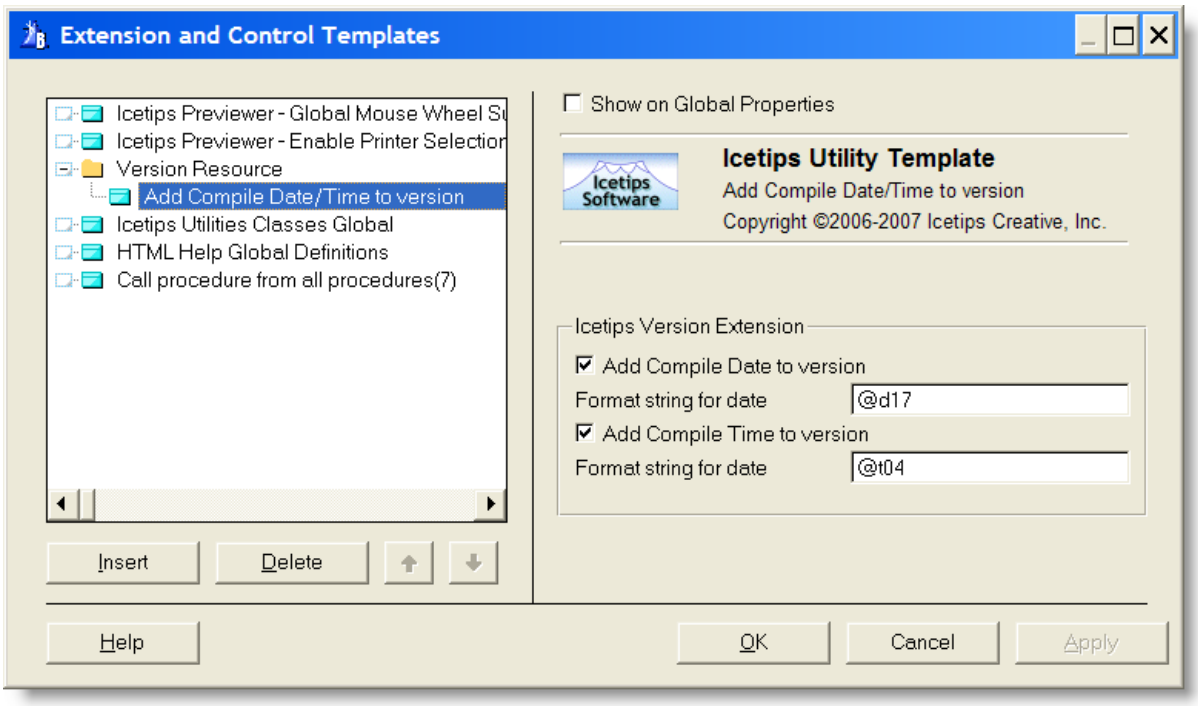
4.3 Extension Templates

4.3.1 Global Extensions Extension Templates

There are currently 13 global extension templates in the Icetips Utilities. Some of them add code to multiple procedures and those should all have an option to quickly and easily exclude procedures from the code being generated by the template.

4.3.1.1 Add Compile Date/Time to version Extension Templates - Global Extensions

This template makes it possible for you to add the compile date and time to the version resources that are linked into the application. This makes it very easy to see exactly when a DLL or EXE was compiled to narrow down problems with client software.



Add Compile Date Check this to add the compile date to the version information

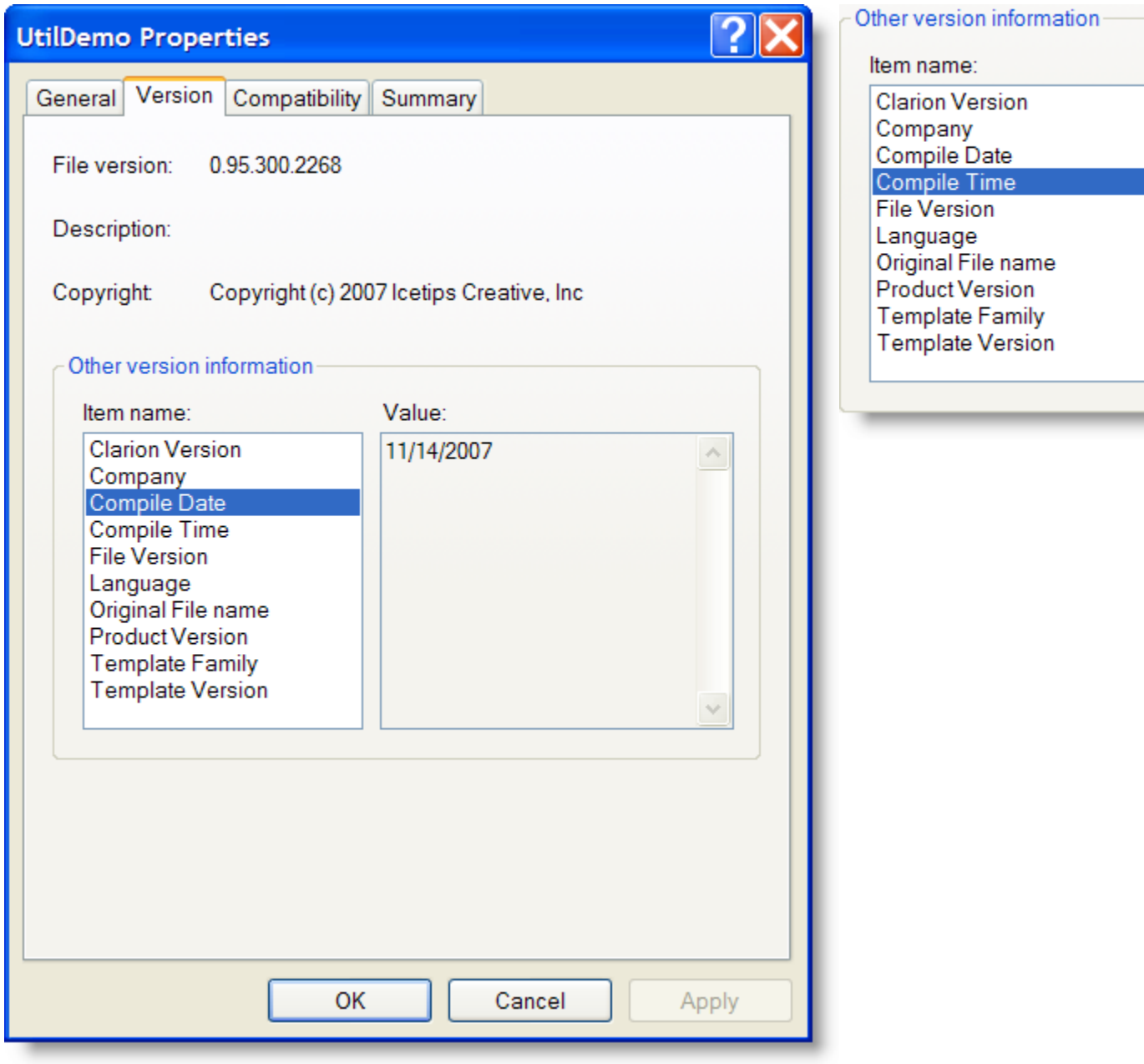
Format string for date Enter the picture for the date. This is used to format the date in the version resource. This defaults to windows short format, @d17.

Add Compile Time Check this to add the compile time to the version information

Format string for time Enter the picture for the time. This is used to format the compile time in the version resource. This defaults to hh:mm:ss format, @t4.

This results in two additional items in the Version information. Below are screenshots that show how this looks for the UtilDemo.exe version 0.95.300 compiled on November 14, 2007 at 12:02:33

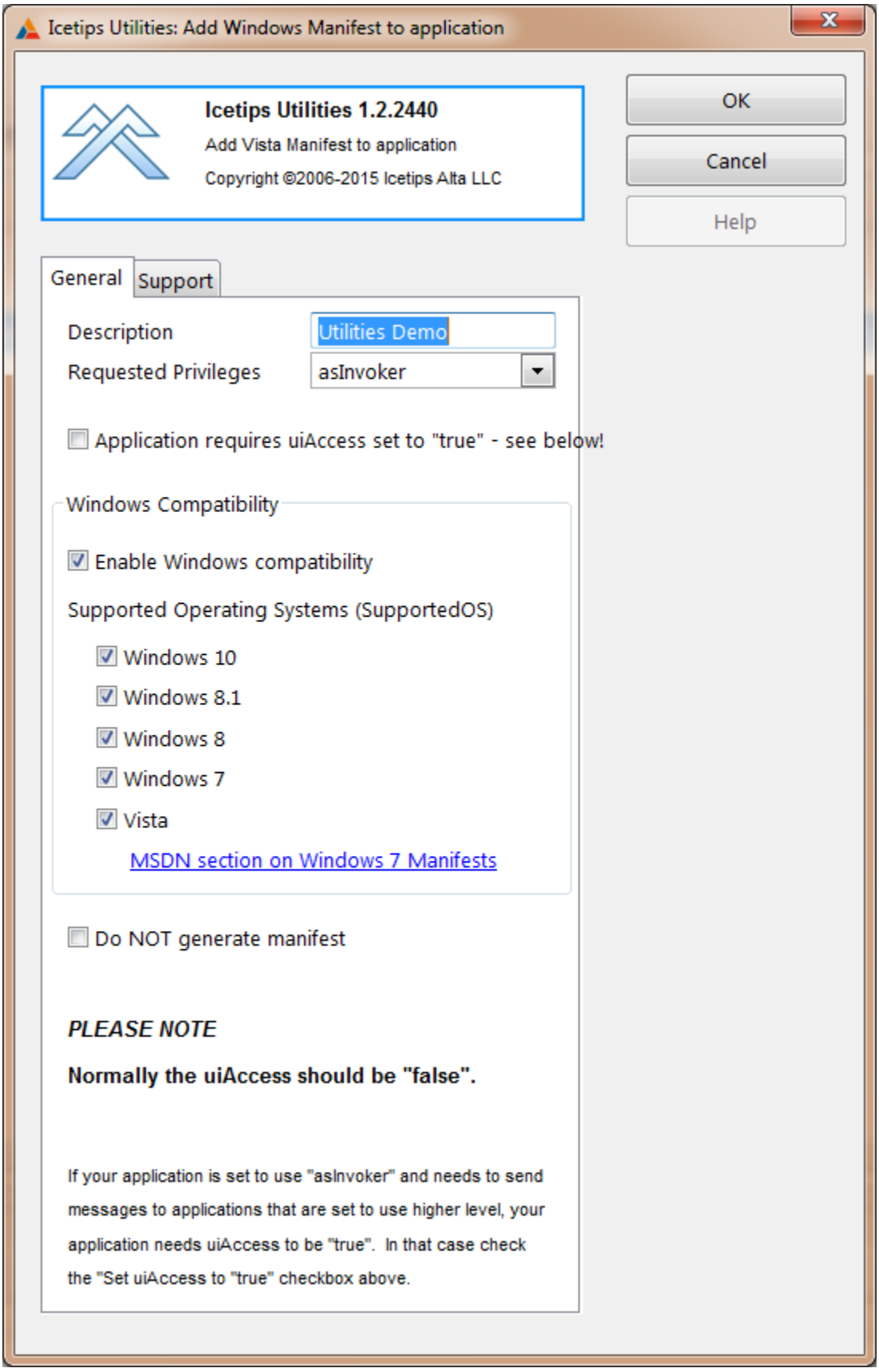
Version information showing the Compile Date **Version information showing the Co**



4.3.1.2 Add Vista/Win7 Manifest to application

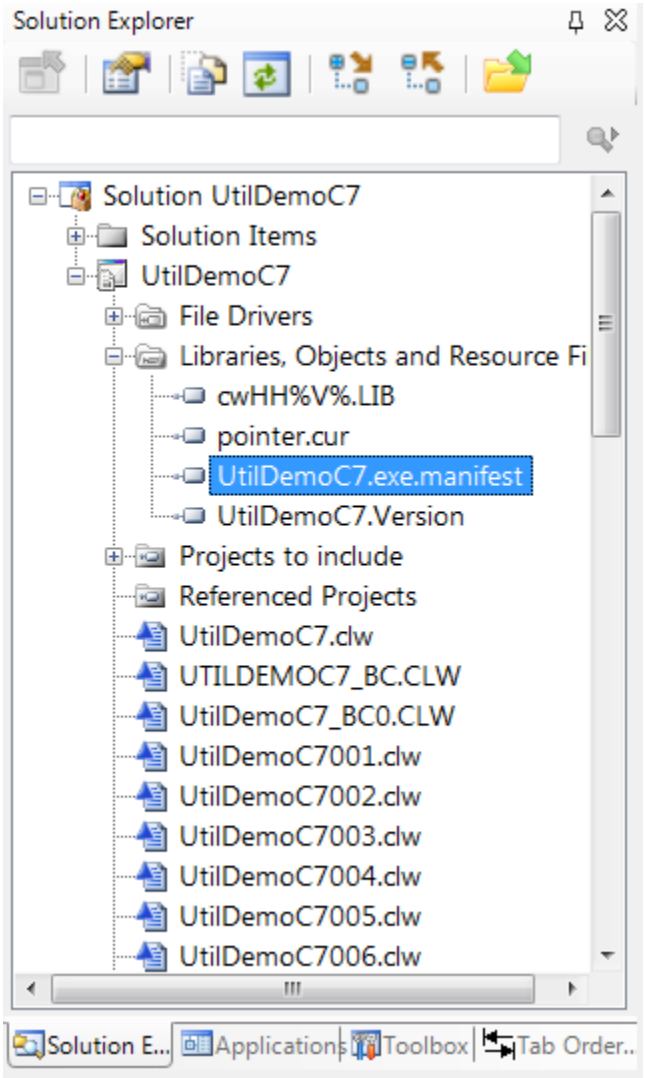
Extension Templates - Global Extensions

This template adds a Vista, Windows 7, Windows 8.0, Windows 8.1 and Windows 10 compatible manifest to your application. This template should be used with all software that may be used on Vista or later computers with User Access Control (UAC) enabled. For more information about UAC please see http://msdn.microsoft.com/en-us/library/aa511445.aspx.



Description	Description of the program. This is put into the <description> tag in the manifest file.
Requested Privileges	This can be one of 3 options: asInvoker, highestAvailable or requireAdministrator. For more information about each of those options please see http://msdn.microsoft.com/en-us/library/bb756929.aspx
App... uiAccess	Normally this checkbox should not be checked. According to MSDN, this setting should only be used for user interface Assistive Technology applications.
Enable Windows compat...	This enables Windows compatibility settings for any version of Windows. Currently it defaults to TRUE. Before November 20, 2015 this defaulted to FALSE.
Supported OS	You can include compatibility setting for any Windows version from Vista onward. If your software will be used on a mix of operating systems, check all that apply. By default they are all set to TRUE. For more information about compatibility mode, please see http://msdn.microsoft.com/en-us/library/dd371711%28VS.85%29.aspx.

The manifest is automatically added to your project. Once you have added the template and generated your project, the manifest will show up in your project in Solution Explorer as shown below.



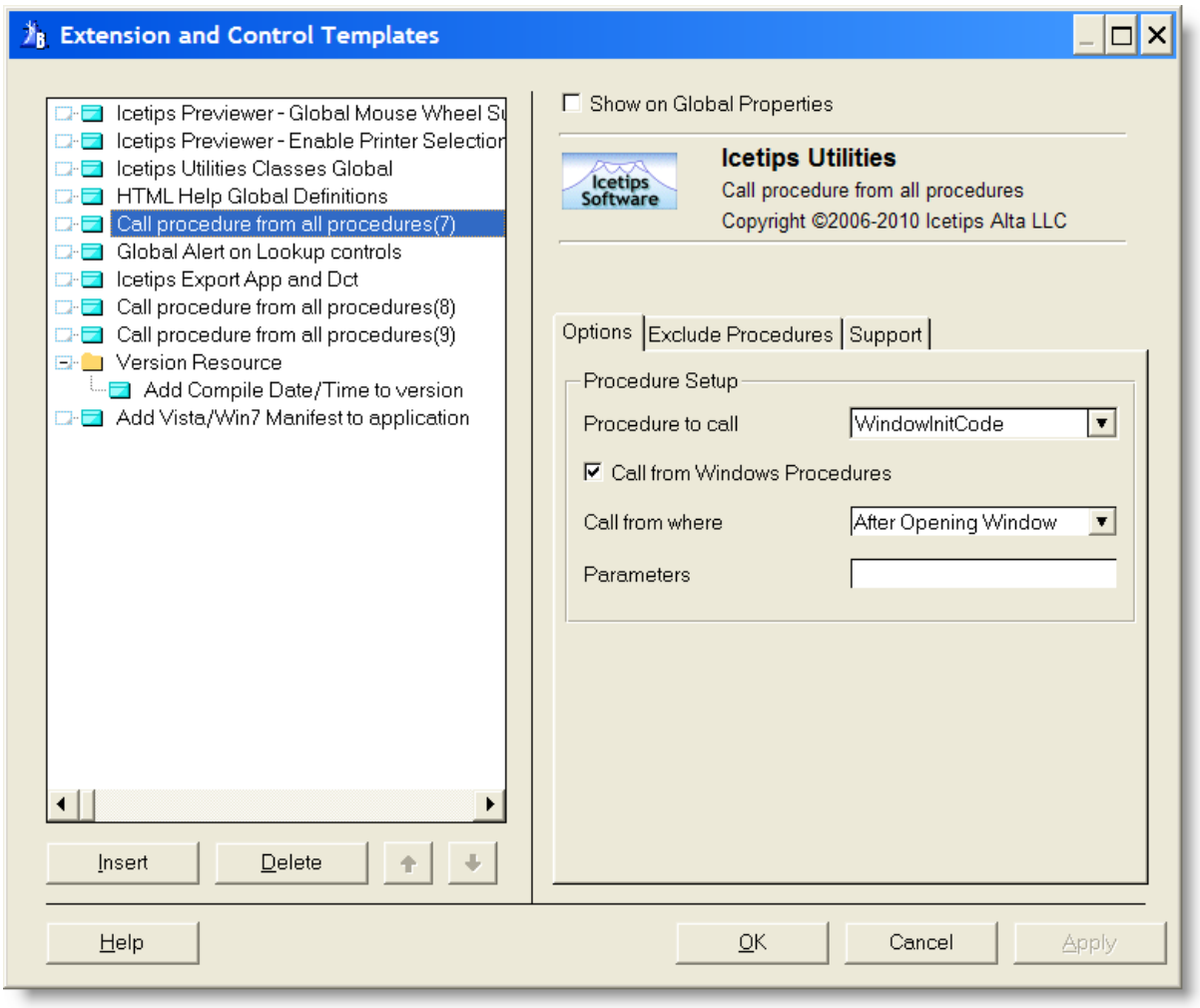
Template and help updated on November 20, 2015

4.3.1.3 Call procedure from all procedures — Extension Templates - Global Extensions

This template allows you to call a single procedure from all procedures in your application. This is extremely useful for procedures that set up windows and controls. For an example of how this is beneficial, check out the WindowInitCode 514 procedure in the UtilDemo.app 514 in your "Clarion\3rdParty\Examples\ITUtilities" folder.

The template has three tabs, Options, Exclude Procedures and Support.

The **Options** tab includes the main options:



Procedure to call Select the procedure that you want to be called from all the procedures in your application except those selected on the "Exlcude Procedure" tab.

Call from Windows... Check this if you only want this to be called from procedures that have a window. This parameter MUST currently be set to true or the template will not generate any code.
Defaults to TRUE (Changed November 29, 2016)

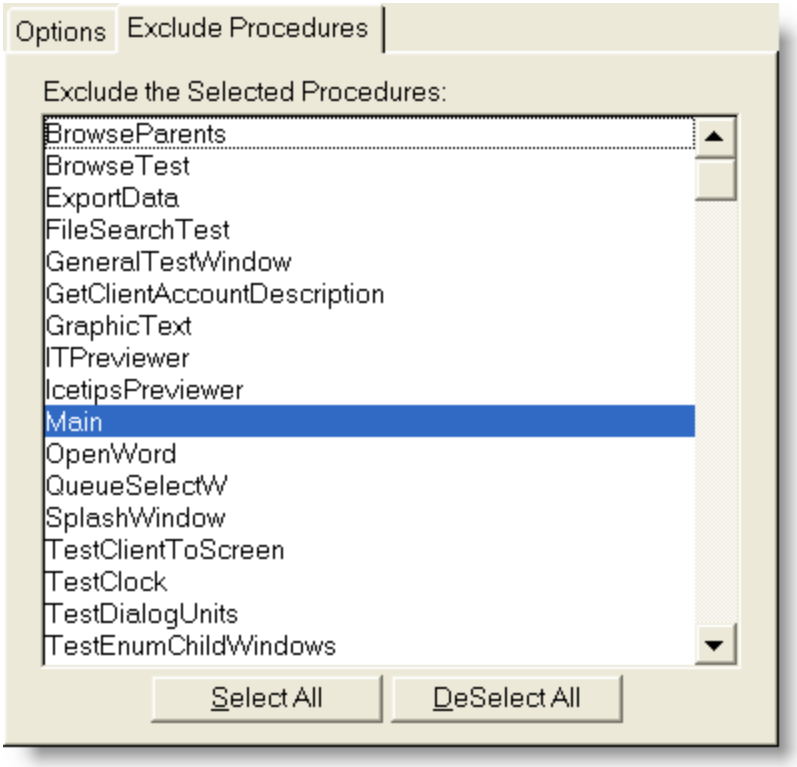
Call from where Select the embed where you want the procedure to be called from. There are 4 options:

Embed	Actual embed location
Start of Procedure	WindowManager.Init, Priority 500
Before Opening Files	WindowManager.Init, Priority 7300
Before Opening Window	WindowManager.Init, Priority 7800
After Opening Window	WindowManager.Init, Priority 8100
After Opening Window - Before Resize	WindowManager.Init, Priority 8110
Window.Init - After Resize	WindowManager.Init, Priority 8260

Parameters Optional parameters to pass to the procedure. This is currently fixed on global

level so it has limited use.

The **Exclude Procedures** tab includes a list where you can select procedures that you want to exclude:

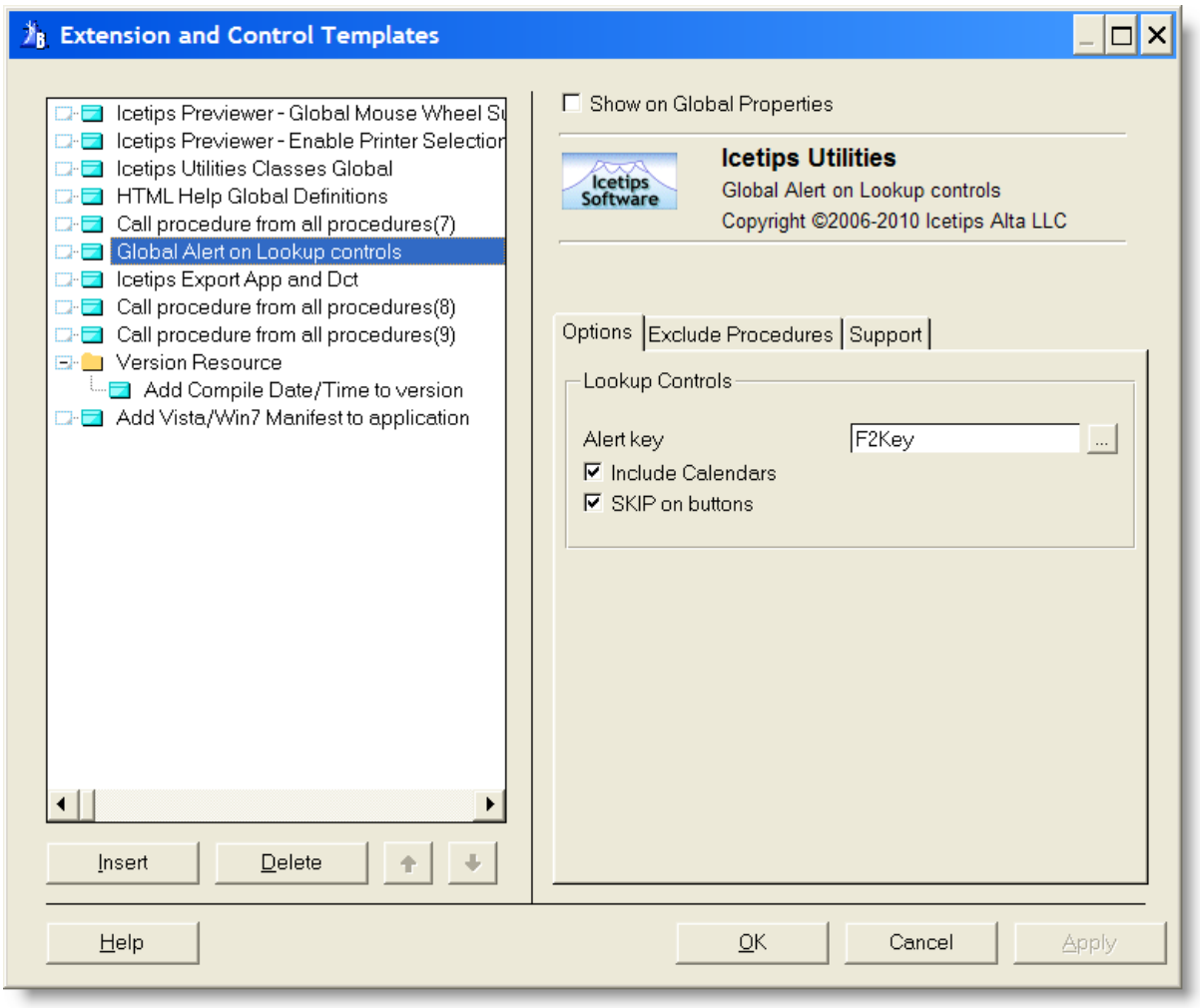


Select the procedures from the list that you do NOT want to call the selected procedure.

4.3.1.4 Global Alert on Lookup controls

Extension Templates - Global Extensions

This template allows you to specify a single alert key that is used on all fields that have a lookup button associated with it or a calendar button. This allows the user of your software to hit a consistent key on the keyboard to bring up lookup browses and calendars.

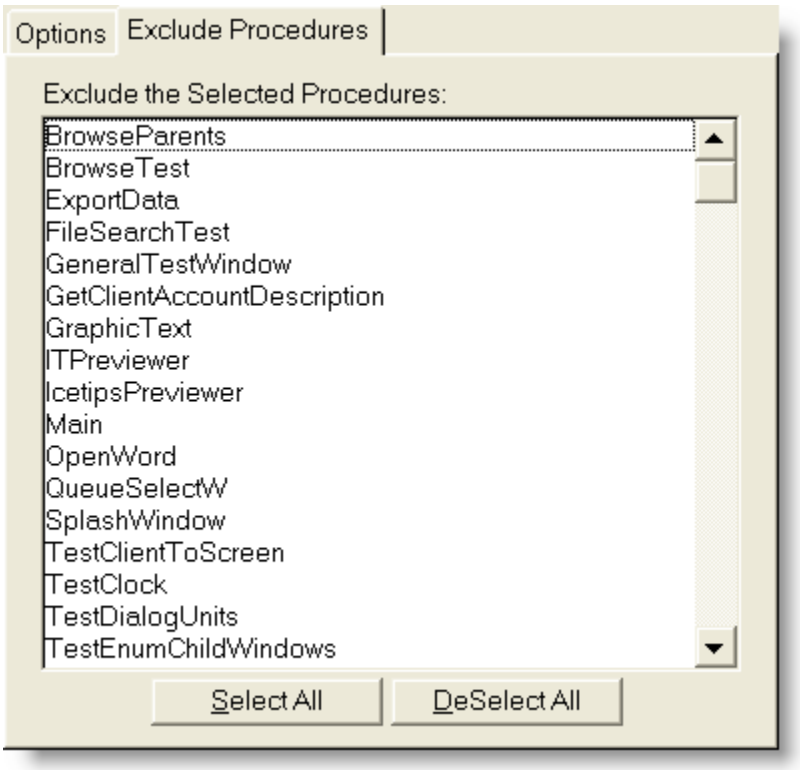


Alert Key — Select the alert key that you want to use. In this screenshot we used the F2 key

Include calendars — Check this if you want to include calendar lookups.

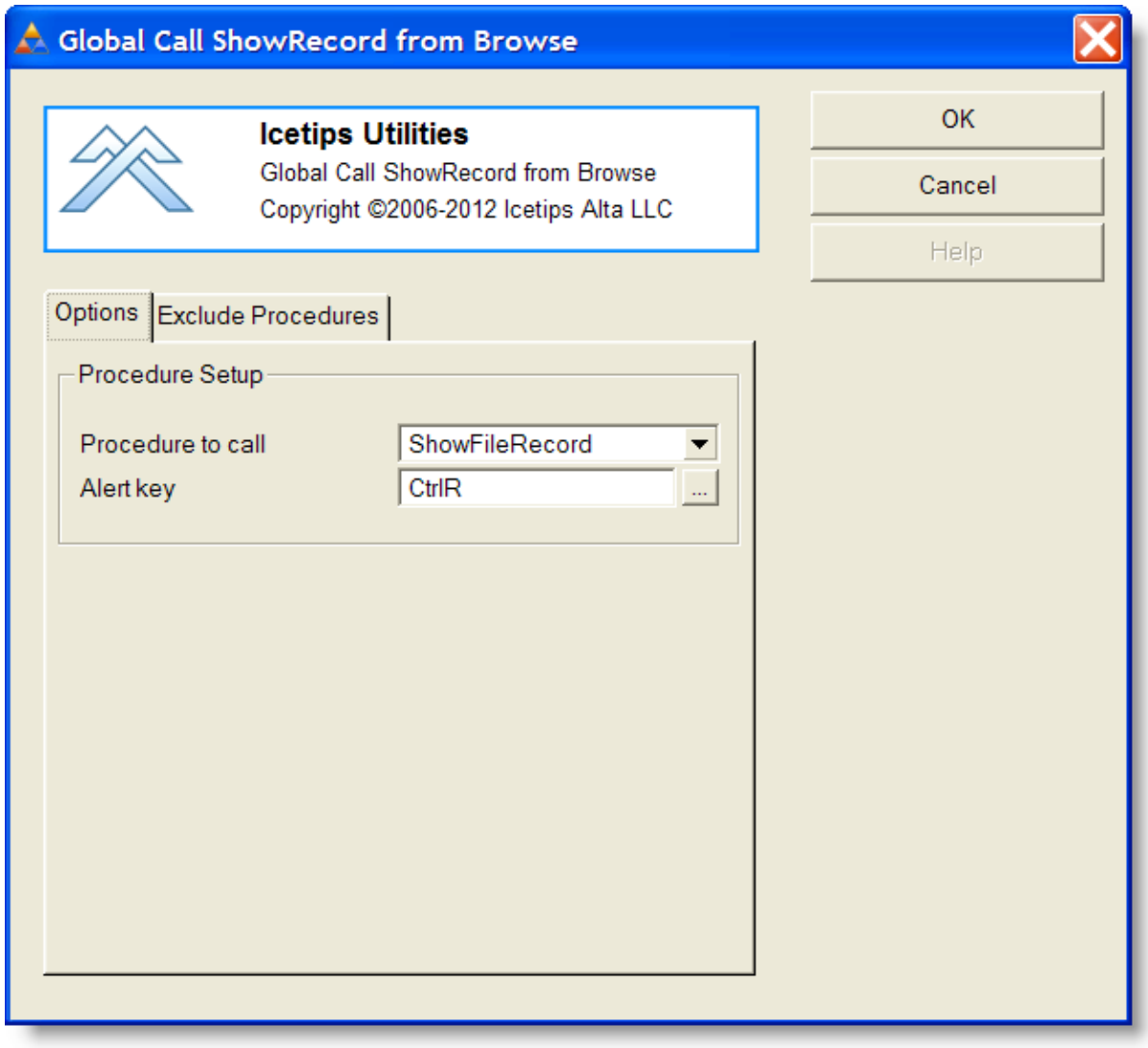
SKIP on buttons — Check this if you want to add the SKIP attribute to the lookup buttons. That way users can use the key on the entry fields and skip the buttons unless they click on them with the mouse, i.e. the Tab key will skip over the buttons.

You can select procedures to exclude fromt he list on the "Exclude Procedures" tab:



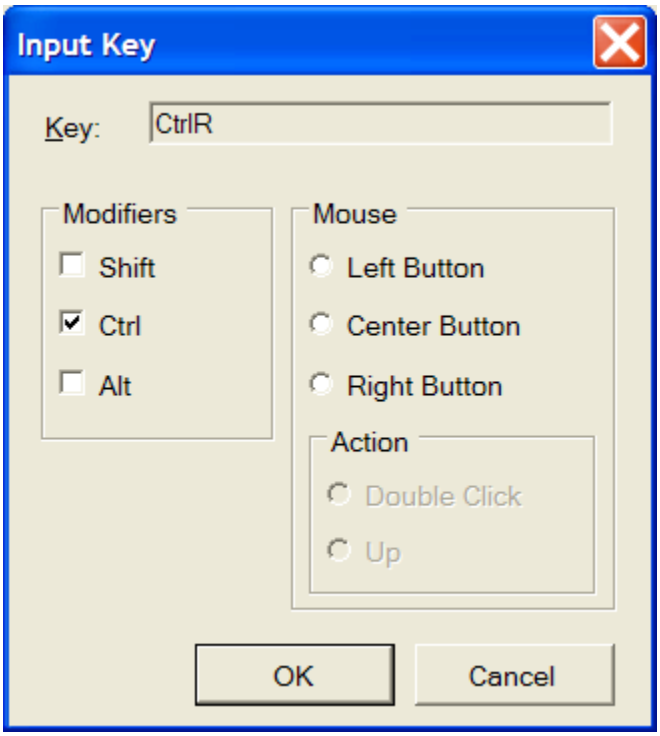
4.3.1.5 Global Call ShowRecord from Browse

This global template adds code to all browse procedures that enables you to call a procedure to view the information from the primary file record. Please see the "Icetips ShowFileRecord Wizard 488" utility template that you can use to create the a procedure. This procedure can be called with a hotkey and will show the information for the file and the file record currently active in the selected browse box.

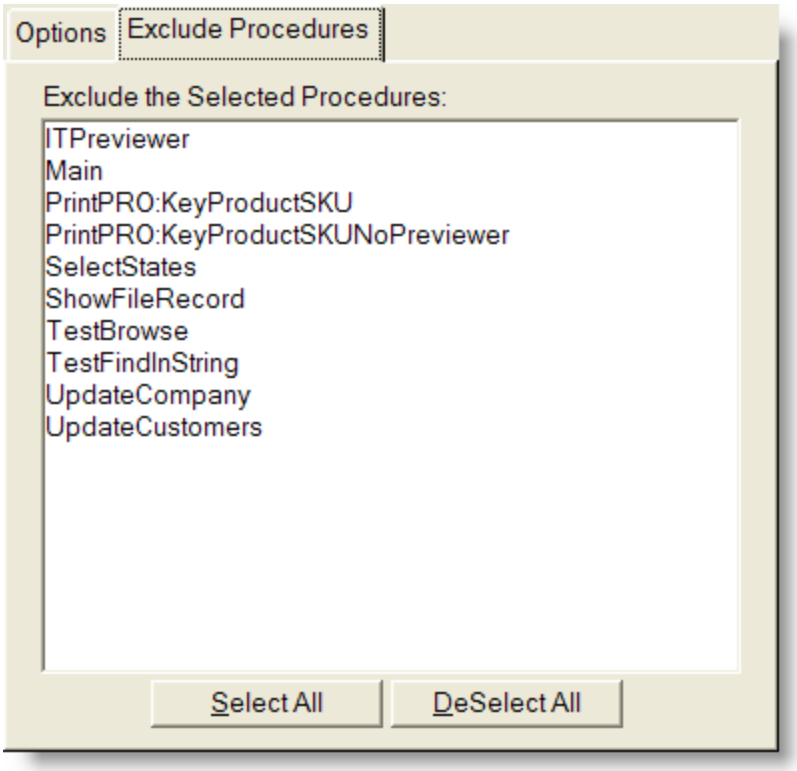


Procedure to call This should be a procedure previously created with the "Icetips ShowFileRecord Wizard" template.

Alert key Select a key that will be used in all browse controls to open the ShowFileRecord procedure specified in the "Procedure to call" dropdown. This key is global to the application. If you click on the [...] button next to the Alert key entry you will get a key selection window that you can use to pick the key. Just hit the combination that you want on the keyboard or select the mouse buttons. You can specify any combination of Shift, Ctrl and Alt keys.



You have the option to quickly and easily exclude procedures from implementing this option. Just selected them on the list on the "Exclude Procedures" tab and the code to call the ShowFileRecord procedure will not be generated.

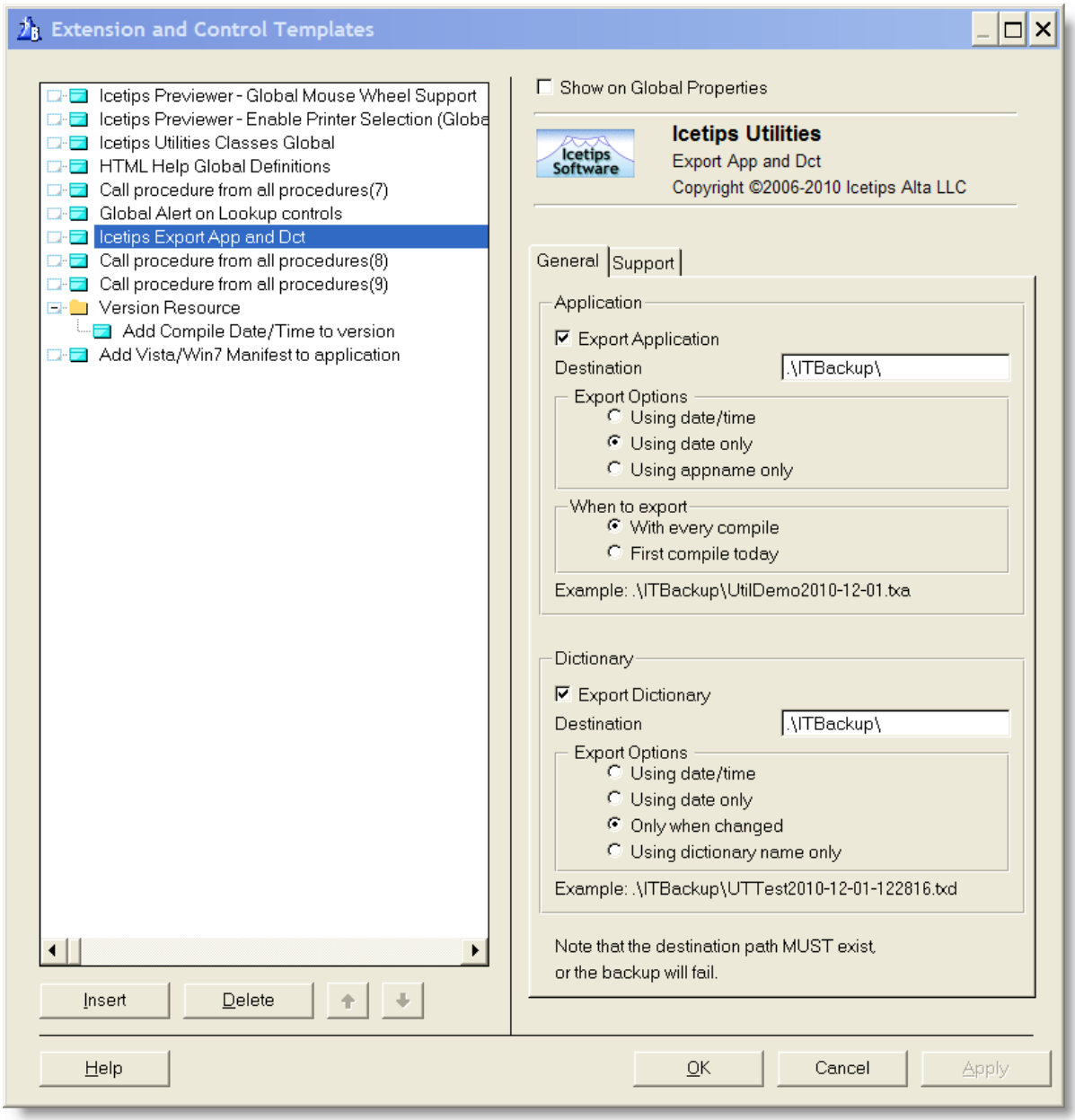


4.3.1.6 Icetips Export App and Dct

This template exports the application and dictionary to TXA and TXD.

NOTE: There have been reports of some issues with TXAs and TXDs generated this way with templates in the past. We strongly recommend that you **DO NOT** rely on those exported files **as your only backup!** The problems that have been reported are in the Clarion export system and it is not up to us to fix.

However, those TXAs and TXDs can be very useful in order to restore, or look at, previous versions of embedded code and table/column information, which is all preserved.



Export Application Check this to export the application to TXA.

Destination The destination folder for the TXA. Note that this folder **MUST EXIST** or the export will fail.

Export Options Determines how to export the app.

- **Using date/time** When using this option a new .TXA is created every time you generate or compile the application.
- **Using date only** This option will only generate one TXA for the day. It is then overwritten when you generate again.
- **Using appname only** When this option is used, there is only one TXA generated and overwritten with each generation.

When to export **Beta 3.3: NOT IMPLEMENTED YET!**
This is used to determine when you want to export the app.

- **With every compile** This will generate the TXA every time you generate or compile the application
- **First compile today** This will generate the TXA only once, the first time you compile it on the current date.

Export Dictionary Check this to export the dictionary to TXD.

Destination The destination folder for the TXD. Note that this folder **MUST EXIST** or the export will fail.

Export Options Determines how to export the dictionary.

- **Using date/time** When using this option a new .TXD is created every time you generate or compile the application.
- **Using date only** This option will only generate one TXD for the day. It is then overwritten when you generate again.
- **Only when changed** When this option is used, the TXD is only generated when the dictionary changes. NOTE: If you use this the dictionary will NOT be exported until you change it. We suggest that you use the "**Using date only**" option, then generate your application to create the first TXD. Then change this back to "**Only when changed**" and then the TXD will be created every time the dictionary changes from now on.

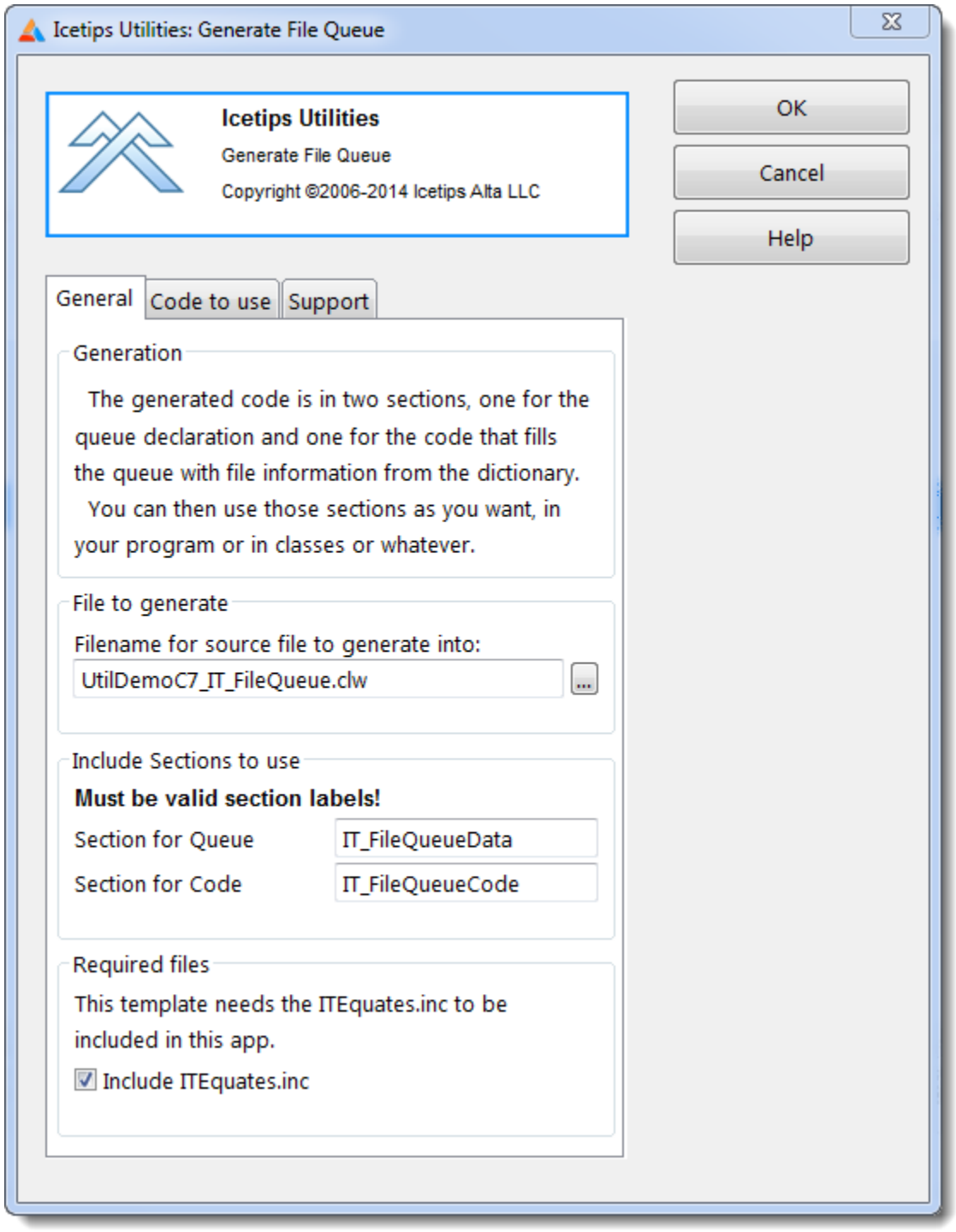
4.3.1.7 Icetips Generate File Queue — Extension Templates - Global Extensions

This extension template generates a code file in two sections, data and code, that creates a queue with information about every file in the dictionary that is used in the application you apply the template to. It gives you access to information that is not otherwise available at runtime, including the file reference and the file manager class reference, label, description, driver, driver parameter, name (i.e.

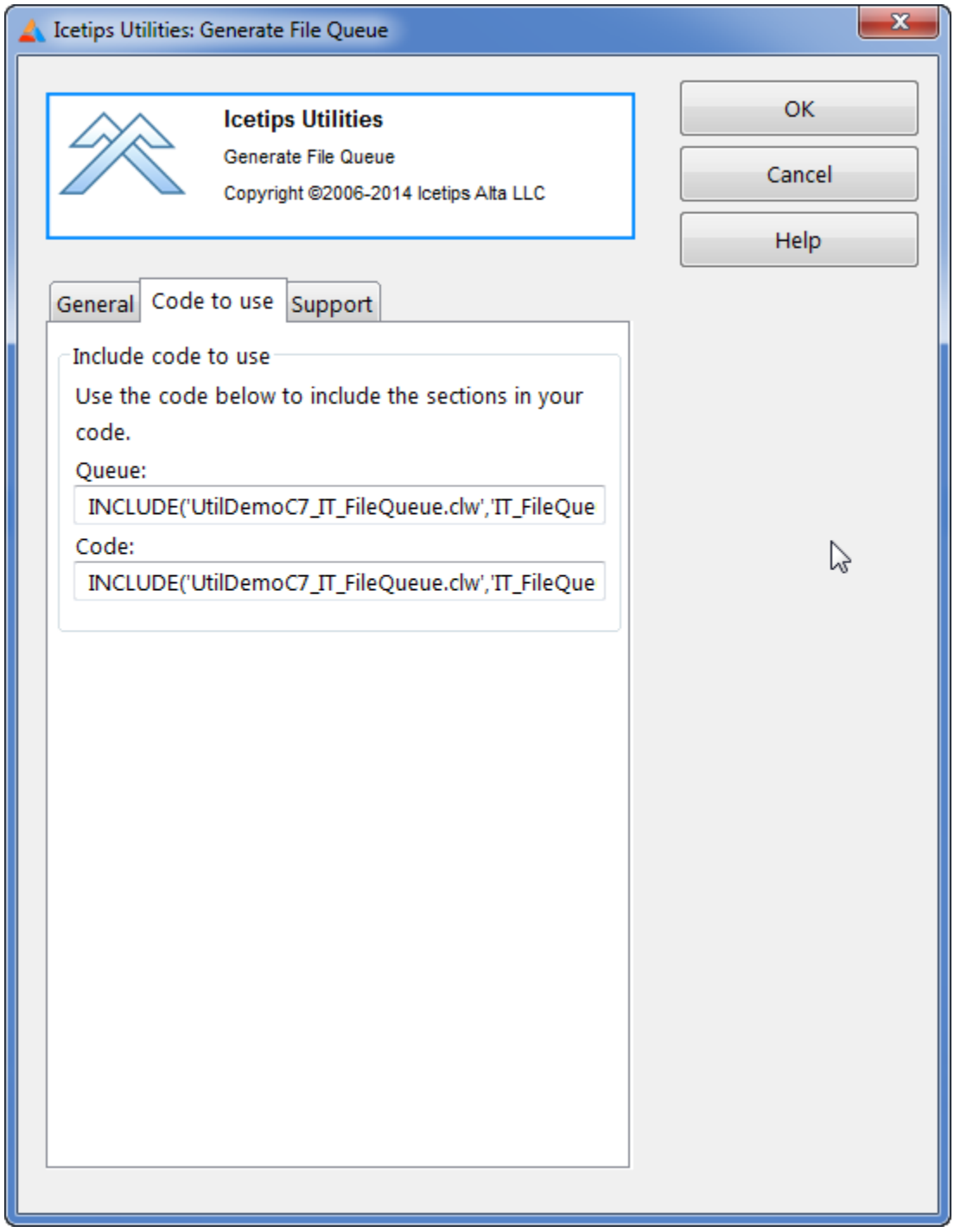
external name), file owner, prefix and type. There will be more information available in this queue later on, such as long description, user options, the file structure, etc. In addition to the file queue I will be adding queues with fields, keys, keyfields, relations etc. so you will have pretty much all the information from the dictionary accessible from your code!

The file queue structure is currently declared in ITEquates.inc as follows:

```
tIT_FileQueue        QUEUE,TYPE
FileRef                &File
FileManager            &FileManager
FileLabel              CSTRING(101)
FileBindable           BYTE
FileCreate             BYTE
FileDescription        CSTRING(1025)
FileDriver             CSTRING(101)
FileDriverParameter    CSTRING(1025)
FileEncrypt            BYTE
FileExternal           BYTE
FileExternalModule     CSTRING(1025)
FileLastModified       CSTRING(30)
FileLongDesc           CSTRING(10241)
FileName               CSTRING(1025)
FileOEM                BYTE
FileOwner              CSTRING(1025)
FilePrefix             CSTRING(21)
FilePrimaryKey         CSTRING(1025)
FileQuickOptions       CSTRING(1025)
FileReclaim            BYTE
FileStatement          CSTRING(1025)
FileStruct             CSTRING(30000)
FileStructRec          CSTRING(30000)
FileThreaded           BYTE
FileType               CSTRING(21)
FileUserOptions        CSTRING(1025)
                     END
```



By default the template will suggest a filename for the generated code file as "<appname>_IT_FileQueue.clw" and sections for the queue structure named as IT_FileQueueData and IT_FileQueueCode. You can change this if you want or keep the default setting.



The "Code to use" tab shows the code that you need to use to include the proper sections in your code. An example code generated from one of the demo dictionaries looks like this:

```
!!
------------------------------------------------------------------------
-
!! 2014-01-27 at 15:14:01       UtilDemoC7_IT_FileQueue.clw
!!
------------------------------------------------------------------------
```

```
-
 SECTION('IT_FileQueueData')
IT_FQ  QUEUE(tIT_FileQueue),PRE(IT_FQ)  !! See ITEquates.inc for the
declaration of this queue
       END

 SECTION('IT_FileQueueCode')
! File Names:
 Clear(IT_FQ)
 IT_FQ.FileRef              &= Names
 IT_FQ.FileManager          &= Access:Names
 IT_FQ.FileLabel             = 'Names'
 IT_FQ.FileDescription       = ''
 IT_FQ.FileDriver            = 'TOPSPEED'
 IT_FQ.FileDriverParameter   = ''
 IT_FQ.FileName              = ''
 IT_FQ.FileOwner             = ''
 IT_FQ.FilePrefix            = 'NAM'
 IT_FQ.FileType              = 'FILE'
 Add(IT_FQ)

! File Products:
 Clear(IT_FQ)
 IT_FQ.FileRef              &= Products
 IT_FQ.FileManager          &= Access:Products
 IT_FQ.FileLabel             = 'Products'
 IT_FQ.FileDescription       = 'Product''s Information'
 IT_FQ.FileDriver            = 'TOPSPEED'
 IT_FQ.FileDriverParameter   = ''
 IT_FQ.FileName              = ''
 IT_FQ.FileOwner             = ''
 IT_FQ.FilePrefix            = 'PRO'
 IT_FQ.FileType              = 'FILE'
 Add(IT_FQ)

! File Parents:
 Clear(IT_FQ)
 IT_FQ.FileRef              &= Parents
 IT_FQ.FileManager          &= Access:Parents
 IT_FQ.FileLabel             = 'Parents'
 IT_FQ.FileDescription       = ''
 IT_FQ.FileDriver            = 'TOPSPEED'
 IT_FQ.FileDriverParameter   = ''
 IT_FQ.FileName              = ''
 IT_FQ.FileOwner             = ''
 IT_FQ.FilePrefix            = 'PAR'
 IT_FQ.FileType              = 'FILE'
 Add(IT_FQ)

! File Children:
 Clear(IT_FQ)
 IT_FQ.FileRef              &= Children
 IT_FQ.FileManager          &= Access:Children
 IT_FQ.FileLabel             = 'Children'
 IT_FQ.FileDescription       = ''
 IT_FQ.FileDriver            = 'TOPSPEED'
 IT_FQ.FileDriverParameter   = ''
 IT_FQ.FileName              = ''
```

```
IT_FQ.FileOwner             = ''
IT_FQ.FilePrefix            = 'CHI'
IT_FQ.FileType              = 'FILE'
Add(IT_FQ)
```

This file is of course updated if your dictionary changes, new files are added or their properties change. This code shows what is currently put into the queue.

To use the code, you simply include it where you need to, the data code in the data section of the procedure or program, and the code in the code section:

```
SomeProc  PROCEDURE
  Include('UtilDemoC7_IT_FileQueue.clw','IT_FileQueueData')
  Code
  Include('UtilDemoC7_IT_FileQueue.clw','IT_FileQueueCode')
```

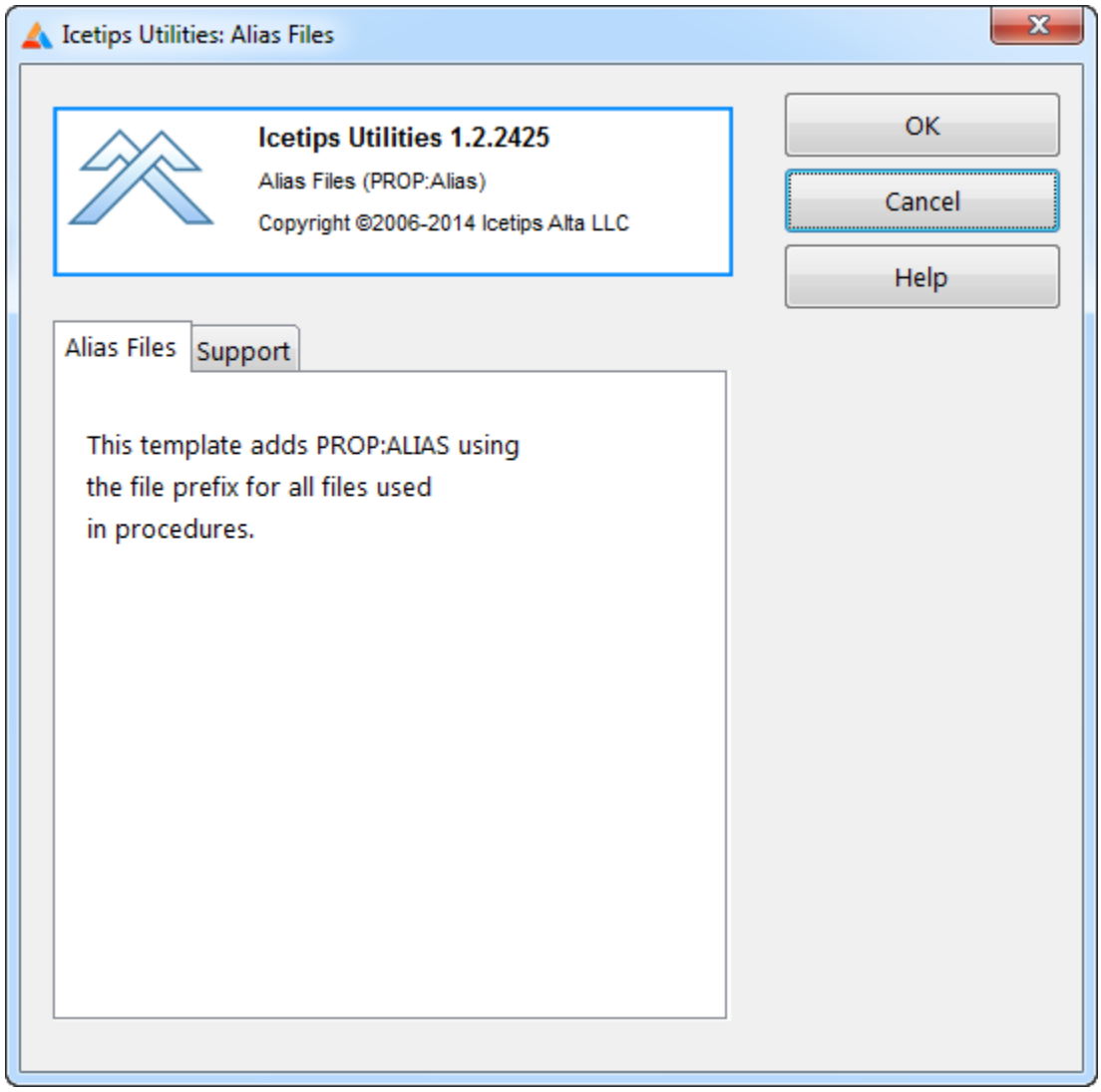
4.3.1.8 Icetips Global Alias Files

Extension Templates - Global Extensions

This extension template adds PROP:Alias for each table that is used in a procedure using the prefix of the table as an alias.

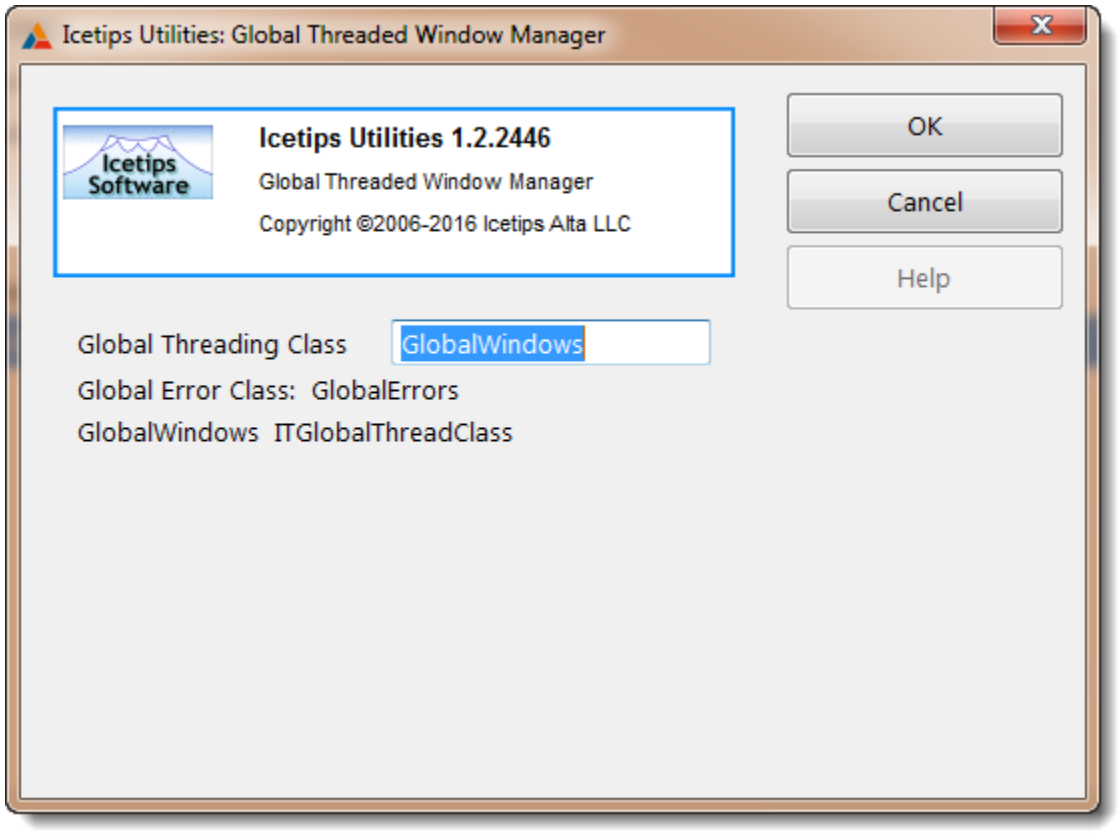
For example if you have a table called MyFile with MYF as prefix, the template will generate:

MyFile {PROP:Alias} = 'MYF'



4.3.1.9 Icetips Global Threaded Window Manager

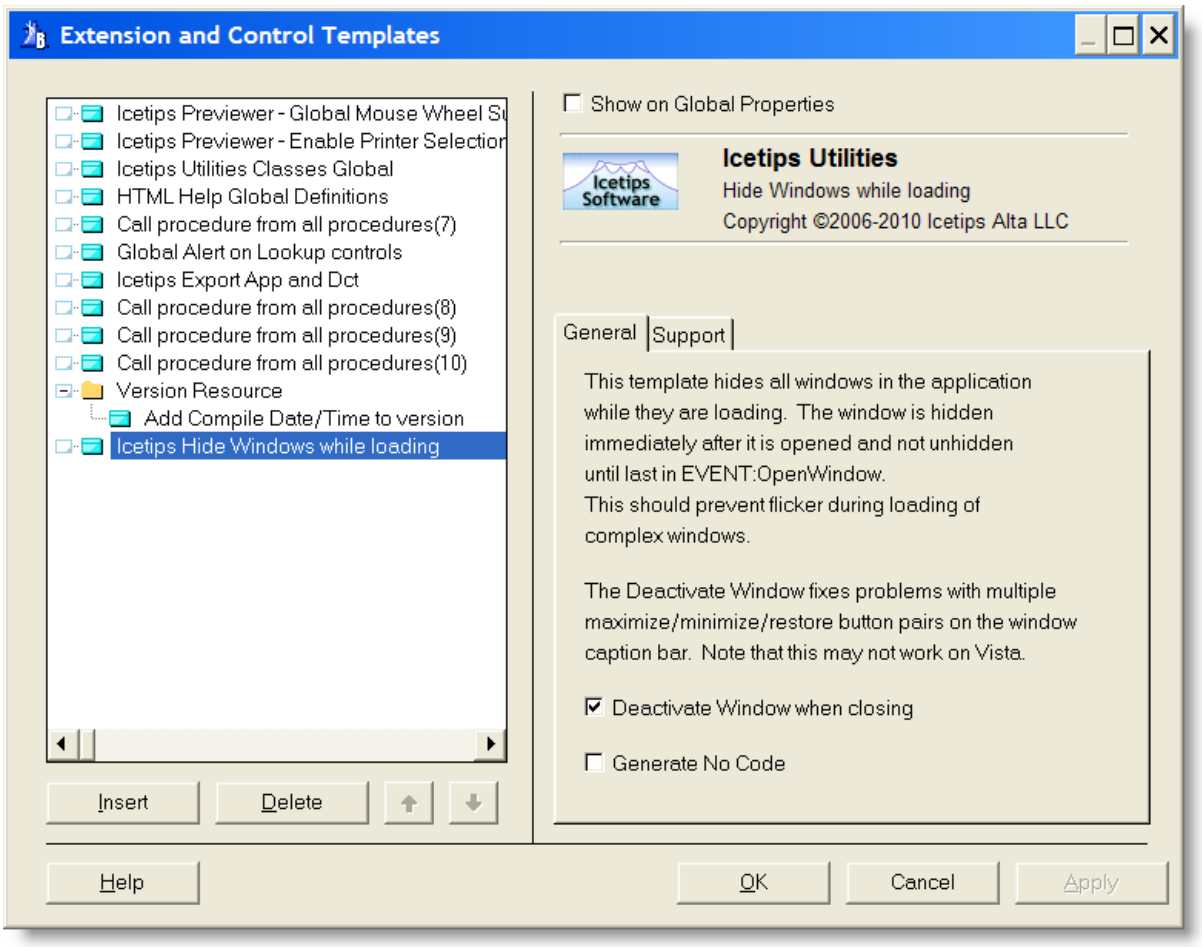
This template is used along with the Icetips Window Manager class 362 to add the global class to the application. The template and class make it possible to close all windows in an ABC application with one command. A procedure template needs to be added to the procedures you wish to interact with the global class. In a multi-dll application this template must be added to ALL your applications including the base/exporting application so the class is declared and exported properly. If that is not done, your application may not compile or it may GPF at runtime.



Global Threading Class The label of the global class. By default it's set to GlobalWindows.

4.3.1.10 Icetips Hide Windows while loading

This template adds code to all window procedures that hides the window while it is loading.



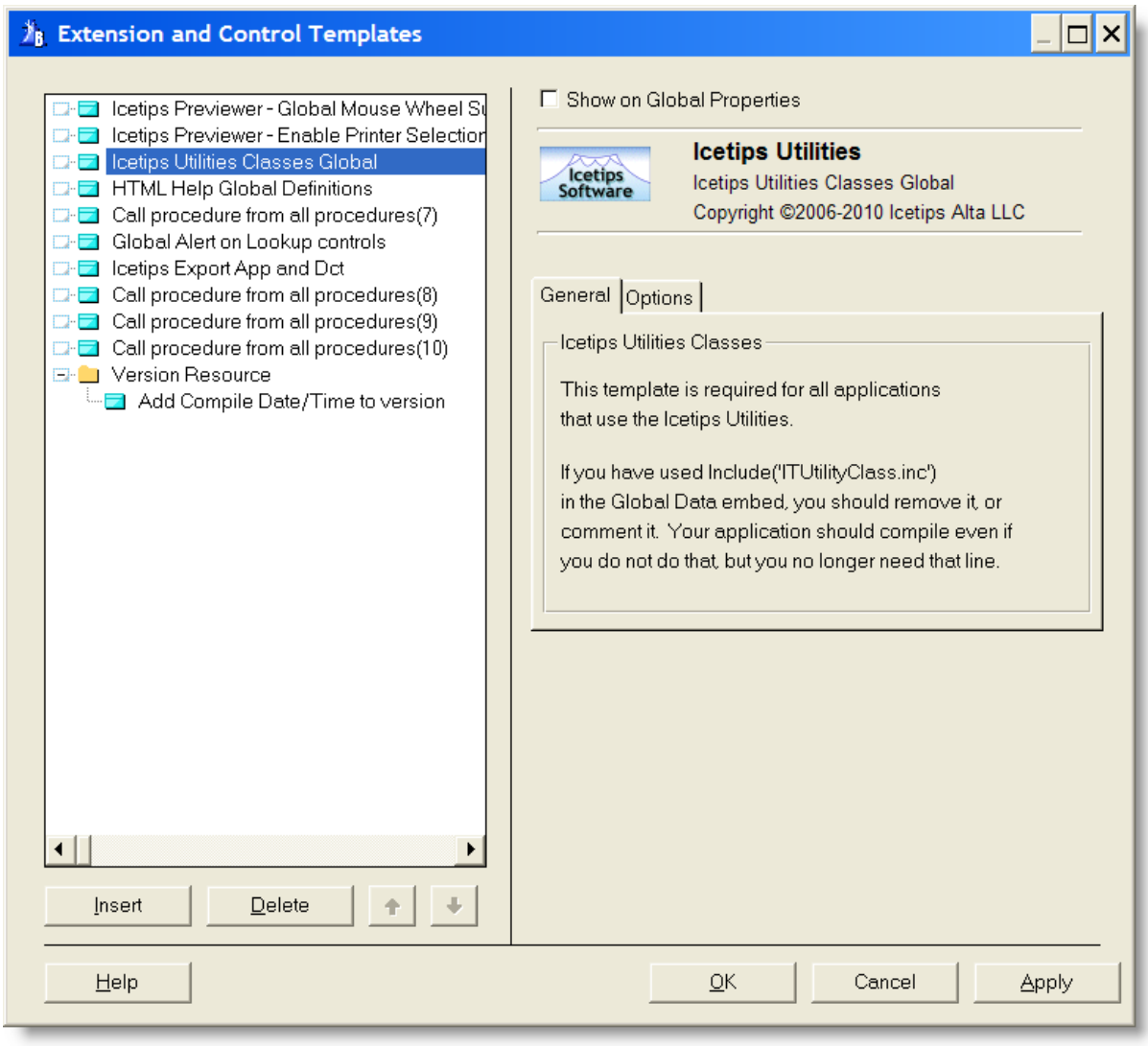
Deactivate Window... This option deactivates the thread when the window closes. I.e. it sets PROP:Active to False for the window.

Generate No Code The template will not generate any code in the application

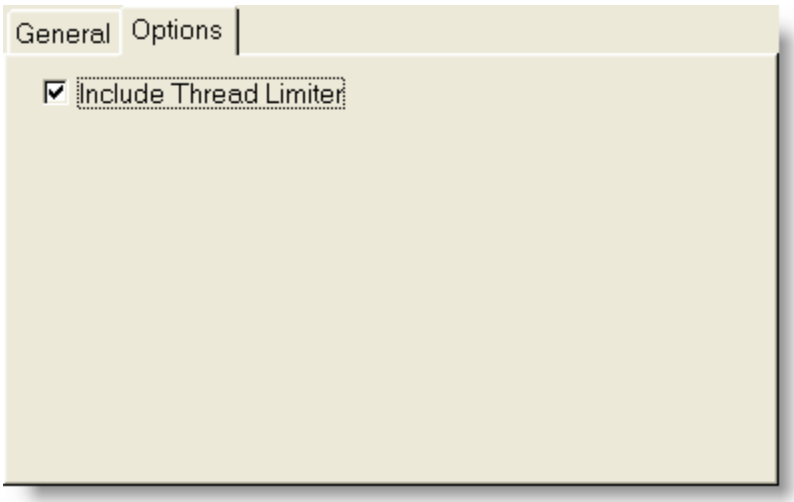
This template hides the window immediately after it opens and sets the mouse cursor to CURSOR:Wait. It then unhides the window in the OpenWindow event handler and turns the wait cursor off. This means that the window will show up fully prepared and without flicker.

4.3.1.11 Icetips Utility Classes Global

This template is used to add the Icetips Utilities Classes to your application. You MUST add this to all applications that need to use the Icetips Utilities Classes. As of build made on June 13, 2012 there are now two different versions of this template, one for Legacy and one for ABC. This allows you to use the Icetips Utilities in Legacy applications as well as ABC. In multi-dll systems this template (both Legacy and ABC) must be applied to the root/exporting dll application as well as any other application that uses any of the Icetips Utilities classes.



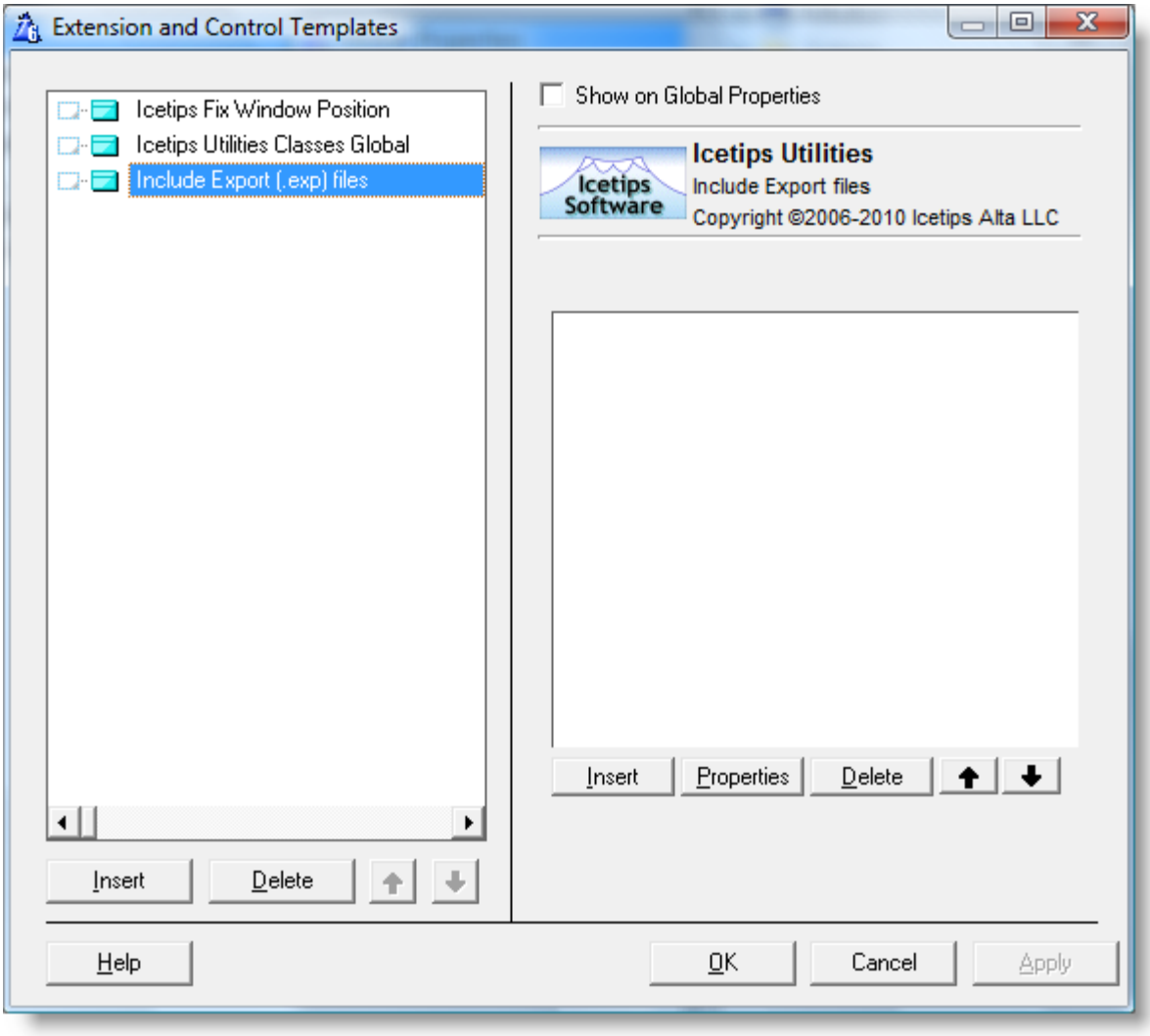
On the Options tab there is currently just one option, to add the Thread Limiter.



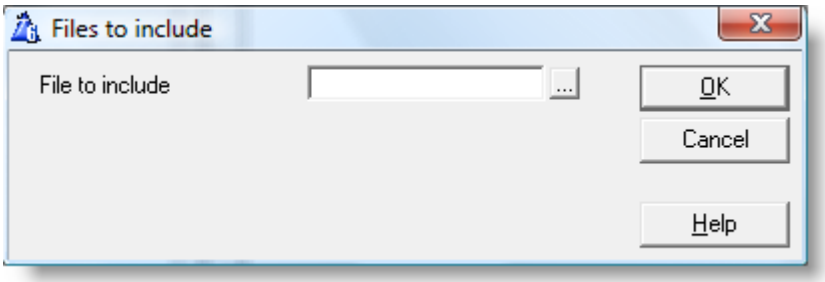
By checking the "Include Thread Limiter" you can now enable thread limiting on your procedures.

4.3.1.12 Include Export files

This template is very useful when there is need to include one or more export files (*.exp) files in an exporting dll.

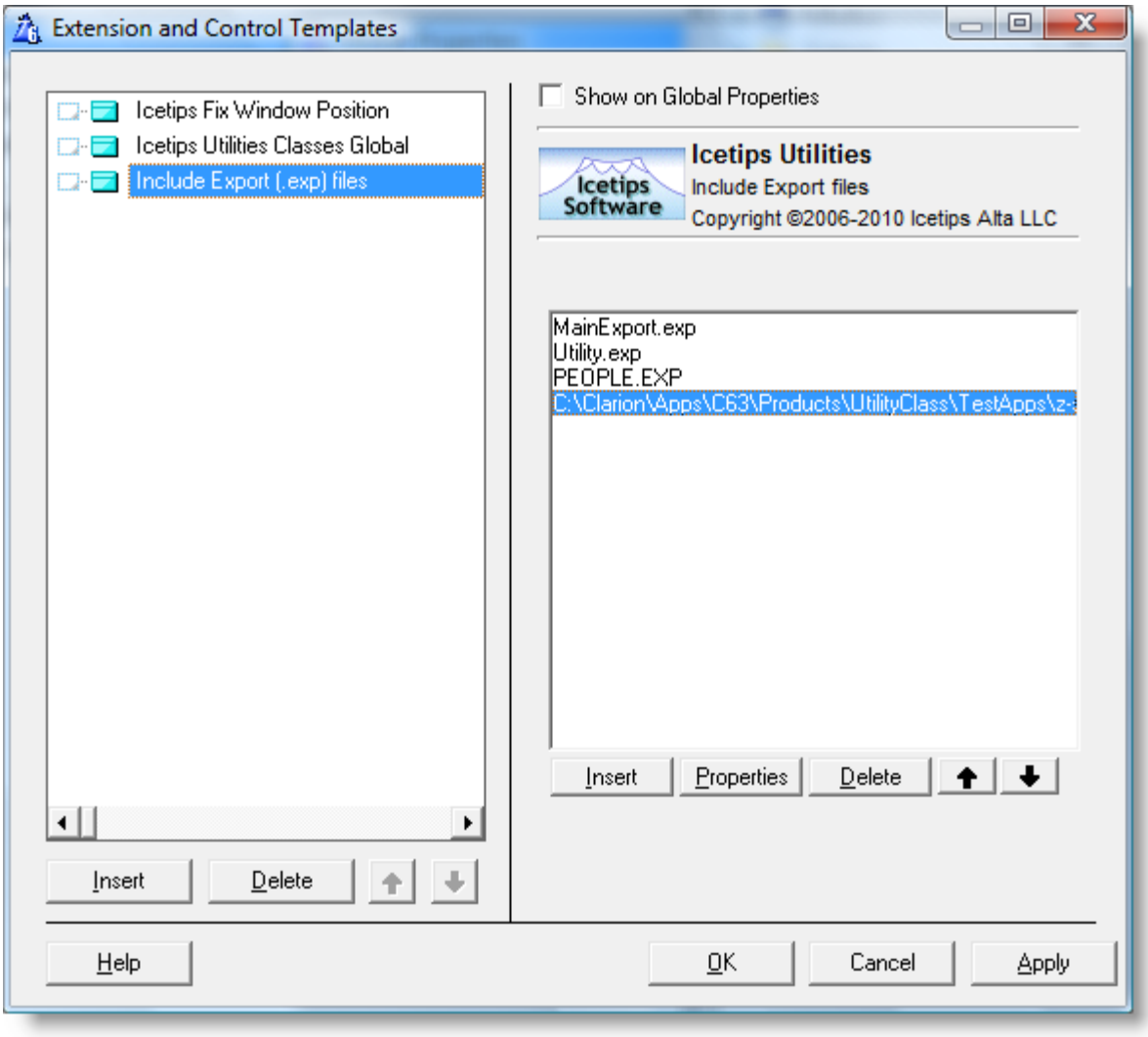


Simply click the "Insert" button to add a .exp file.



Click the [...] button to select the .exp file. If it is in the same folder as the application or can be found

via the REDirection file, the filename will be added without the path. It the file cannot be located with the RED file, it will be imported with full path.



When the application is generated, those 4 .exp files will be generated into the application .exp file. The exp section will be generated starting with a comment, like this one:

```
; ------------------------------------------------------------------------------
; Included by Icetips "Include Export (.exp) files" template on February 16, 2010 at 10:57:49
; ------------------------------------------------------------------------------
```

Then a comment for each file that is being read in and included, like this one:

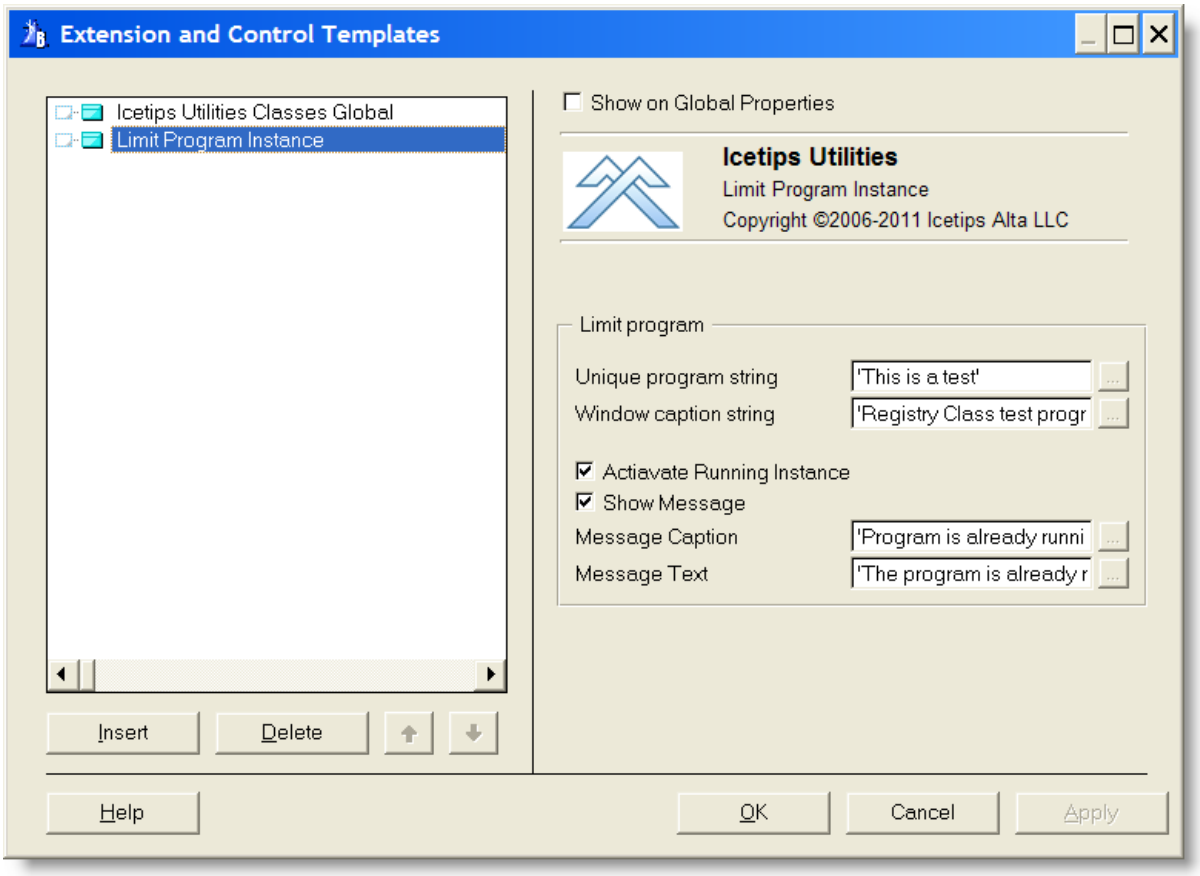
```
; Included file: MainExport.exp
```

And then a list of the exports. This is repeated for each included file. This makes it very easy to find this generated section in the generated exp file.

4.3.1.13 Limit Program Instance

Extension Templates - Global Extensions

This templates makes it easy to prevent multiple instances of the program from executing at the same time.



Unique Program String

This is a string that you create and is unique to this program. You could use the Core Class Demo to create a GUID if you want to use a string that you can be pretty safe will be unique. Just remember that once you assign it to the program you should never change it if you want to make 100% sure that your program will always be unique and never run more than one instance on any computer. If you need to run multiple instances of the program, but each with a different setting (for example different command line parameters) you can use a variable for this and set it to something unique for each instance. Because this code executes right after the program CODE statement, you could use a simple source function to get the unique string. Here is an example of a very simple source procedure:

```
GetProgramID          PROCEDURE  (String pID)!!,String
S    CString(100)
 Code
 S = Clip(pID)
 If Command(1)
   S = S & Command(1)
 End
 Return(S)
```

To use this, you put in:

```
! GetProgramID('MyProgram')
```

to the "Unique program string" entry. Note the exclamation mark at the beginning. It is required to make the template interpret it as not being a string constant. You could of course also let the GetProgramID() procedure construct

the whole string and just use GetProgramID() to get the string.

Window caption string	This string is used to find the running instance on the computer and then activate it.
Activate Running...	Check this if you want to activate the running instance of the program when the user tries to run it again.
Show Message	Check this if you want to show a message when a second instance is found.
Message Caption	Caption for the message to display.
Message Text	Text for the message to display.

Note that you can use variables for any of the text entries by using ! in front of the name. Anything else will be interpreted as a literal string. You do not have to put single quotes around string literals, the template will do that automatically when generating the code.

The code is generated right after the CODE statement for the program. There are 5 embed points created by this template in a tree node called "Icetips | Limit Program Instance" so you can add code before and after the generated code, as well as before the Message(), before the window is activated and before the HALT() so you do have some flexibility. You can also use the method described above for the Unique Program String and use that function to do any program initialization you want to complete before the program validation is performed.

See also:
IsProgramRunning 398
Activate Window 378

4.3.1.14 Write Template info to file

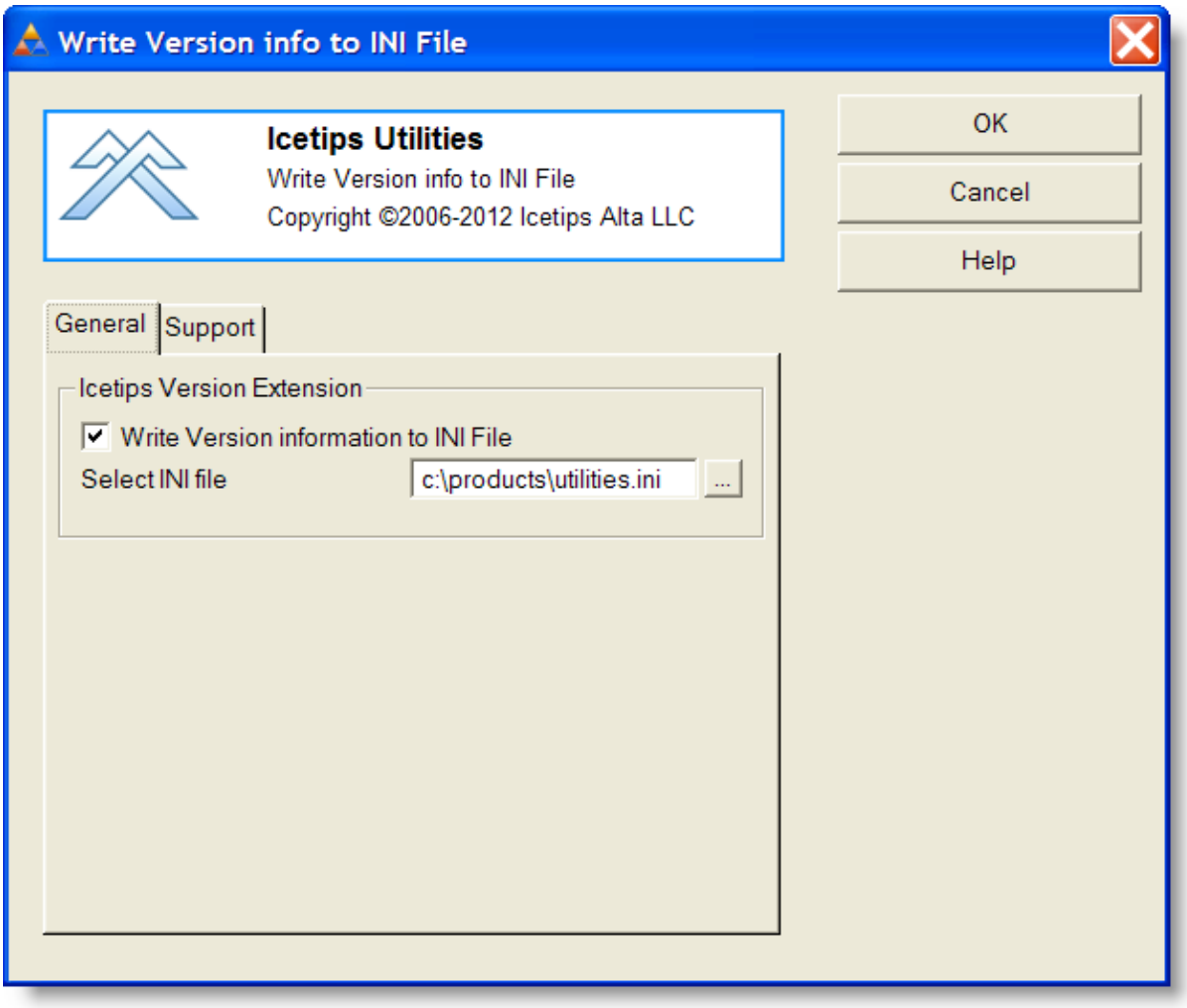
Extension Templates - Global Extensions

This template writes a list of the global extension templates to a file with a .tpx extension. Note that the Write Templates To File utility template 498 has replaced this template. This extension has been deprecated and may be removed in future versions (as of June 22, 2012, I'm keeping this extension in the utilities but generally the utility template should be used)

4.3.1.15 Write Version info to INI File

Extension Templates - Global Extensions

This template requires the "Version Resource" template from Softvelocity and before you try to add this template you need to select the "Version Resource" template. This template writes all the version information from the "Version Resource" template to an ini file that you pick. This can come in handy with any kind of automation for example if you are using Build Automator but you can't let it handle the version information for whatever reason. Then you can export the version information and use the "Get From INI" action in Build Automator to retrieve the build information.



Write Version info... If checked, the template will write the version information to the file specified below

Select INI file Select the INI file that you want to write to. Please note that the path to it MUST exist and the template will not attempt to validate or create the path to the INI file.

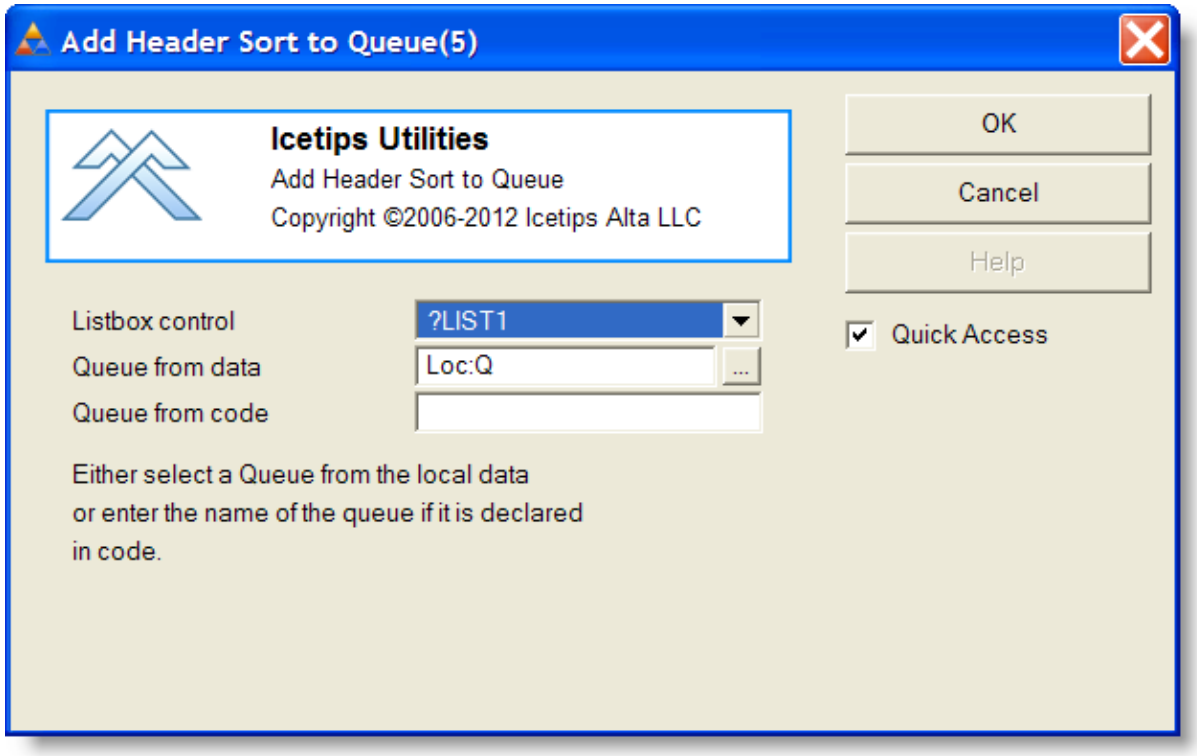
When the application is generated, it will write an INI file that will look like this:

```
[VersionInfo]
Application=TestPageOfPages.app
Application Path=C:\Dev\Prod\Utilities\Apps\C8\DemoApp\
Product Name=Page of Pages Demo app
File Description=
ProductVersion=1.2.1234.3
FileVersion=1.2.1234.3
ProductMajorVersion=1
ProductMinorVersion=2
ProductSubVersion=1234
ProductBuildNumber=3
FileMajorVersion=1
FileMinorVersion=2
FileSubVersion=1234
FileBuildNumber=3
```

4.3.2 Procedure Extensions Extension Templates

4.3.2.1 Add Header Sort to Queue Extension Templates - Procedure Extensions

This template adds the header sort option to a regular queue driven listbox, just as it does to Clarion browse templates. This is a very handy way to create listboxes that the user can sort by clicking on the header.

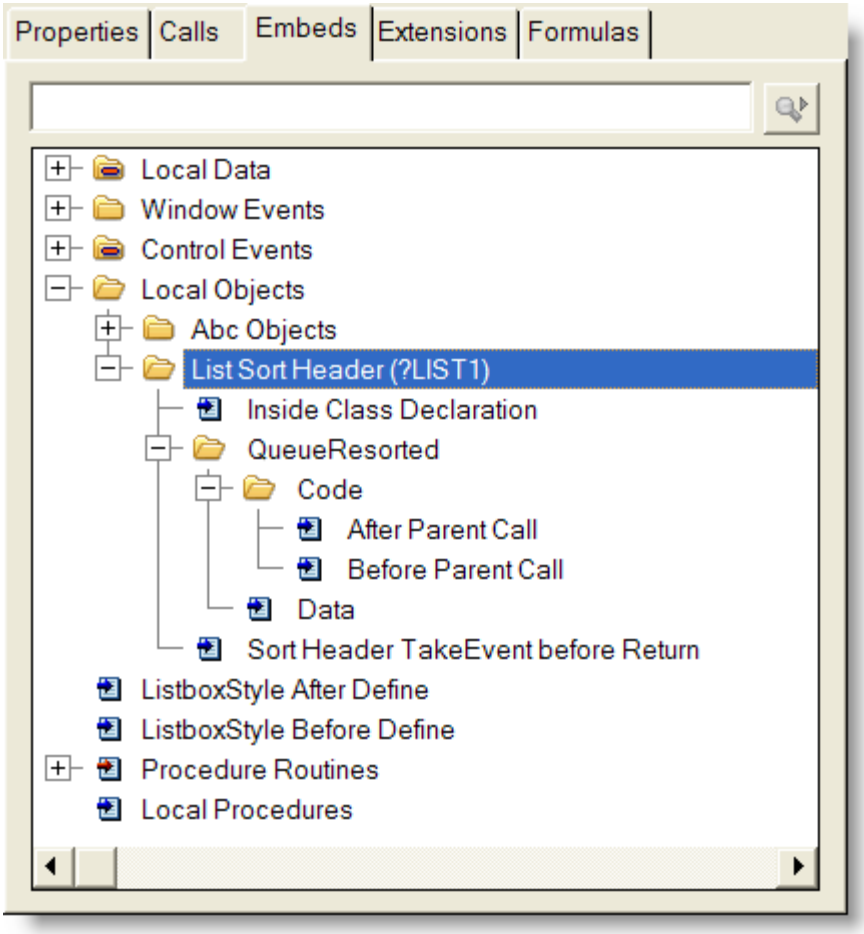


Listbox control The listbox control to add the header sort to.

Queue from data Use this option to select a queue from the data declaration in the app

Queue from code Use this option to enter a queue label that is declared in embedded code rather than in the data.

This template adds some embeds to the procedure, see the screenshot below.



Inside Class Decl.

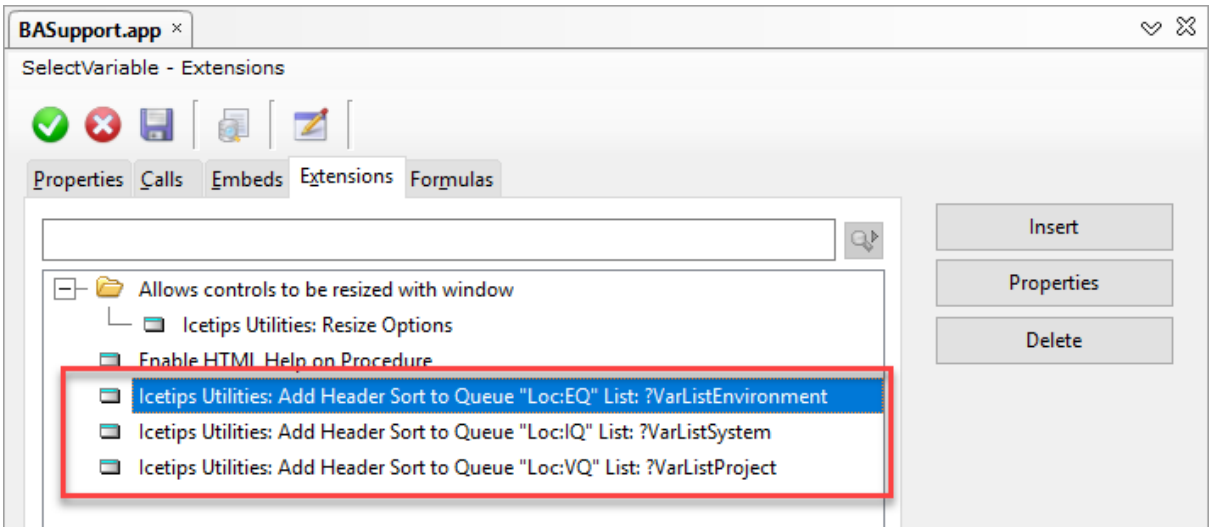
This embed is inside the CLASS(SortHeaderClassType)/END structure that declares the derived sort header class. This embed can be used to declare inherited methods if needed. Note that the QueueResorted method is always declared by the template.

```
ITSQ5::SH CLASS(SortHeaderClassType) !Declare SortHeader
Class
! Start of "Inside Sort Header Class"
! [Priority 4000]
! ***  This is the "Inside Class Declaration" Embed.
! End of "Inside Sort Header Class"
QueueResorted          PROCEDURE(String pSortingString),
VIRTUAL
                       END
```

QueueResorted

This embed can be used to execute code that should run after the queue is sorted. There is a bug in the SortHeaderClassType.QueueResorted, which results in the queue not having been sorted when this method is called. The only way around it was to let our installer change the PRIVATE attribute on the ListQueue property in the brwext.inc file so I could sort the queue inside the derived QueueResorted method. The results are that in the "After Parent Call" embed the queue is sorted correctly.

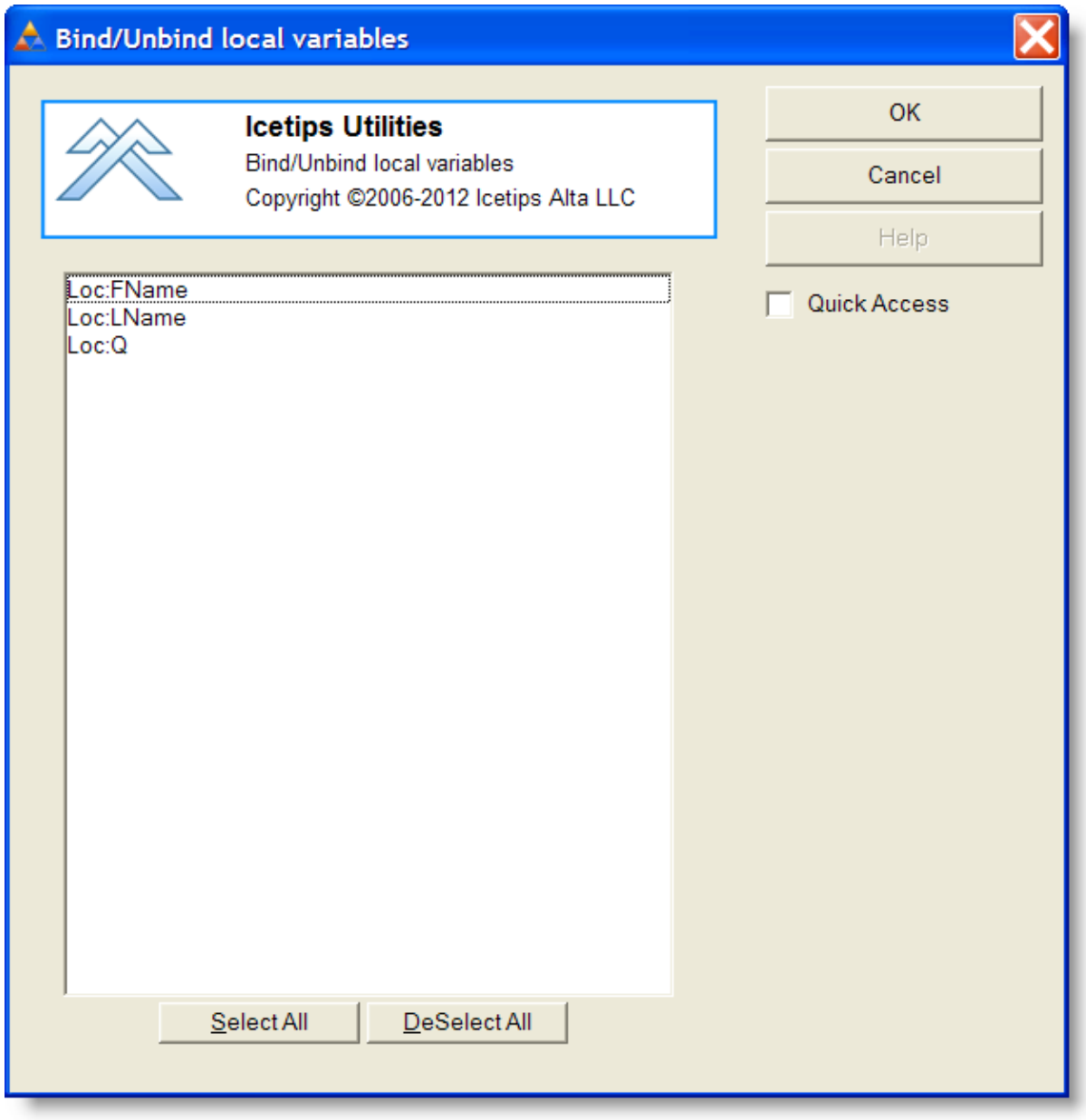
The extension list shows which queue and list are involved, which helps to identify which extension affects what list/queue:



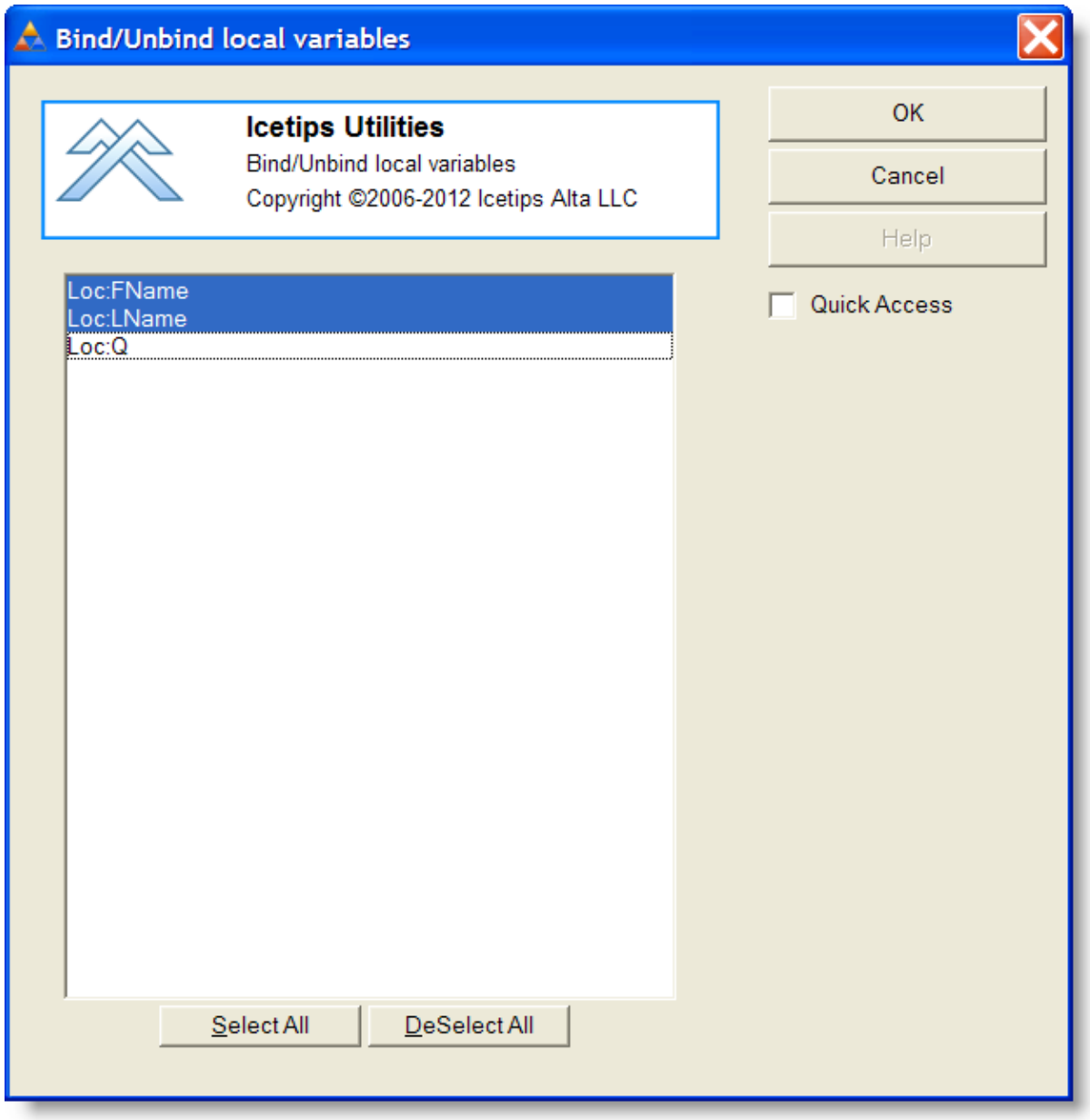
4.3.2.2 Bind/Unbind local variables

Extension Templates - Procedure Extensions

This template allows you to very quickly BIND local variables where you need to, for example if you need to use them in PROP:Filter or with EVALUATE. It simply gives you a list of all local variables and you select which ones you want to BIND.



To select, you just need to click on the entries. You can also use the "Select All" and "DeSelect All" buttons at the bottom to quickly select or deselect all the variables in the list. Here is an example of how it looks like with two variables selected.



This selection will result in this code being generated into the ThisWindow.Init method code section at priority 5501:

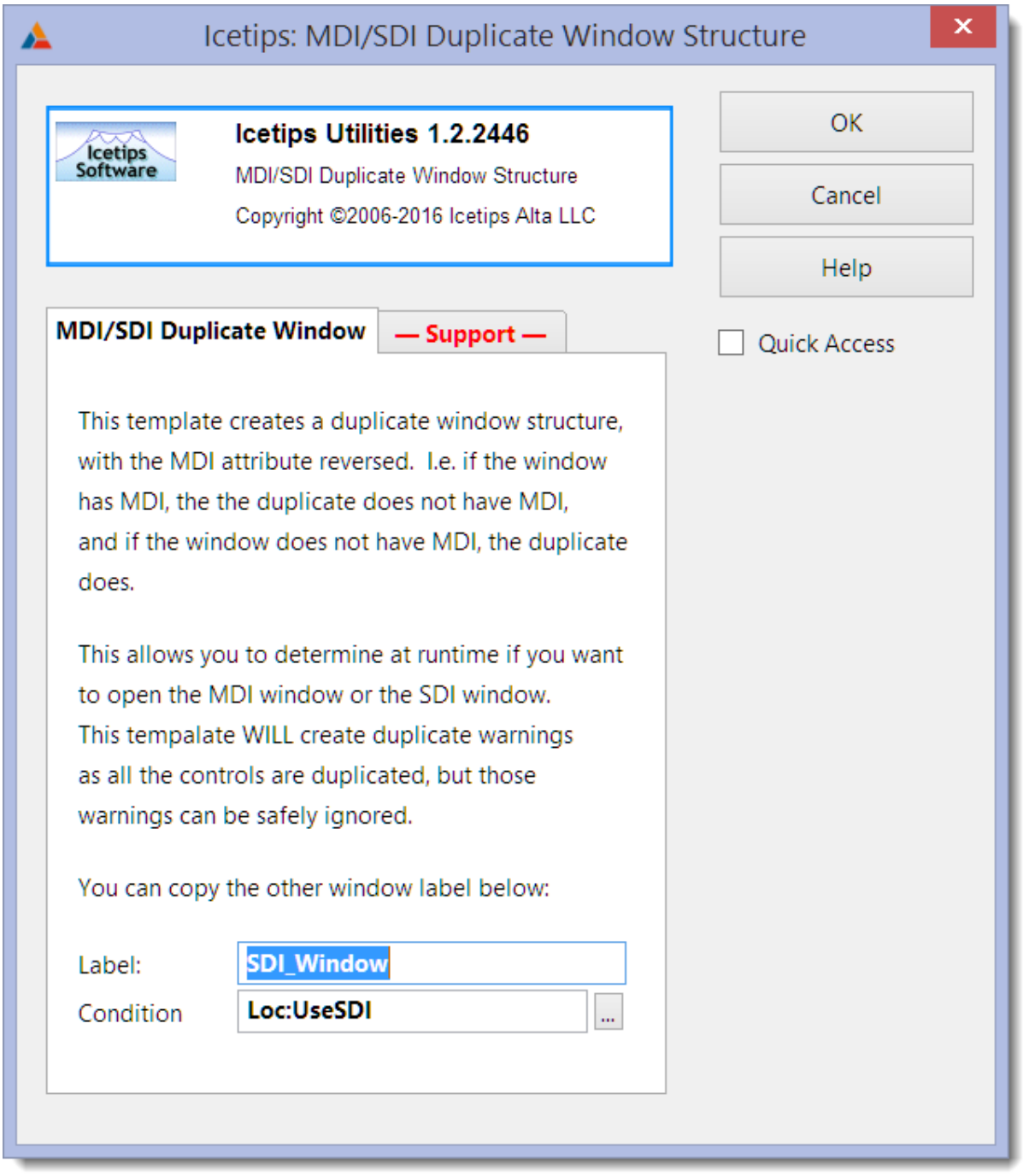
```
    BIND('Loc:FName',Loc:FName)                          ! Icetips Bind/Unbind
local variables
    BIND('Loc:LName',Loc:LName)                          ! Icetips Bind/Unbind
local variables
```

And the corresponding UNBIND code generated into the ThisWindow.Kill method code section at priority 4501:

```
    UNBIND('Loc:FName')                                  ! Icetips Bind/Unbind
local variables
    UNBIND('Loc:LName')                                  ! Icetips Bind/Unbind
local variables
```

4.3.2.3 Duplicate Window

This template has one purpose: To duplicate a window structure so the window can be opened as either a SDI window or MDI window. This makes it possible to use the same window procedure from the application frame procedure as a MDI window as well as before the application frame is opened as a SDI window.



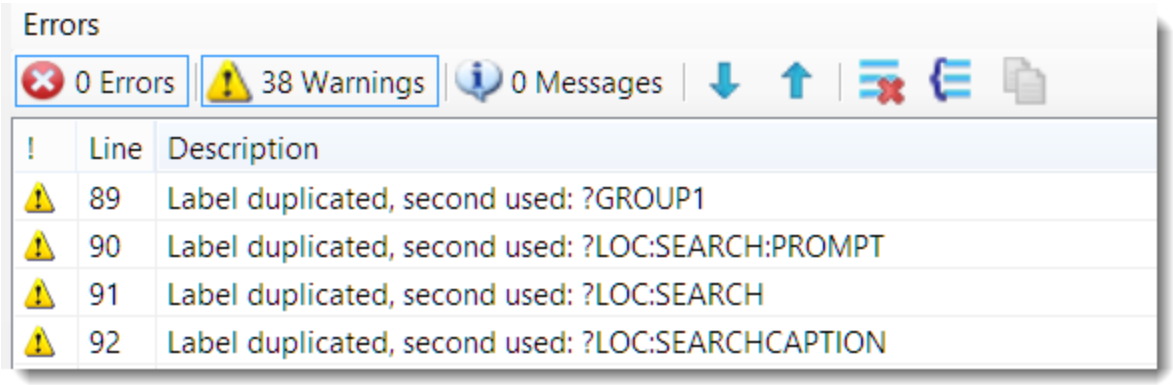
Label This shows the label of the duplicate window. Note that this field is read-only and the window label cannot be changed. The label is the actual window label, prefixed with either SDI_ or MDI_

Condition The condition to switch. If TRUE then what ever is the duplicated window is used. In this case the actual window is a MDI window so the duplicated window is a SDI window

This will generate code that conditionally opens the SDI or MDI windows as shown below:

```
386   ! [Priority 7700]
387
388     IF Loc:UseSDI
389       SELF.Open(SDI_Window)
390     ELSE
391   ! [Priority 7999]
392
393   ! Open the window
394   SELF.Open(Window)                                    ! Open window
395     END
396   ! [Priority 8001]
```

Note that during generation, this template WILL cause duplicate label warnings as shown below for every control that is on the window as they are indeed duplicated in the duplicated window. Those warnings can be safely ignored.

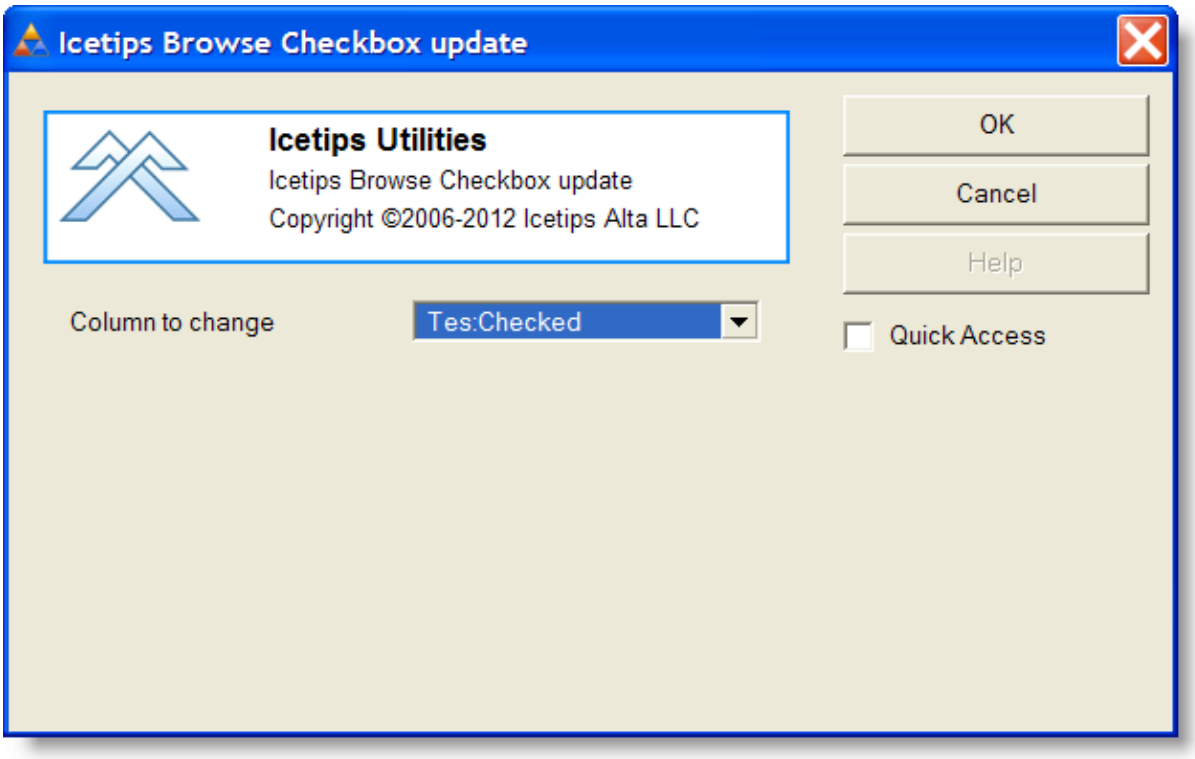


4.3.2.4 Icetips Browse Checkbox update

Extension Templates - Procedure Extensions

This is a very simple template that allows you to add a checkbox update with a double click on a browse that does not use edit-in-place. Double clicking on the cell with an icon set will flip the true/false value of the data field. Note that this template works with ABC browses only and is currently limited to one column, but you can add multiple instances of the template to the browse to handle multiple columns.

Note that the ABC browse must be selected before you click on the "Insert" button or this template will not show up in the available extensions.

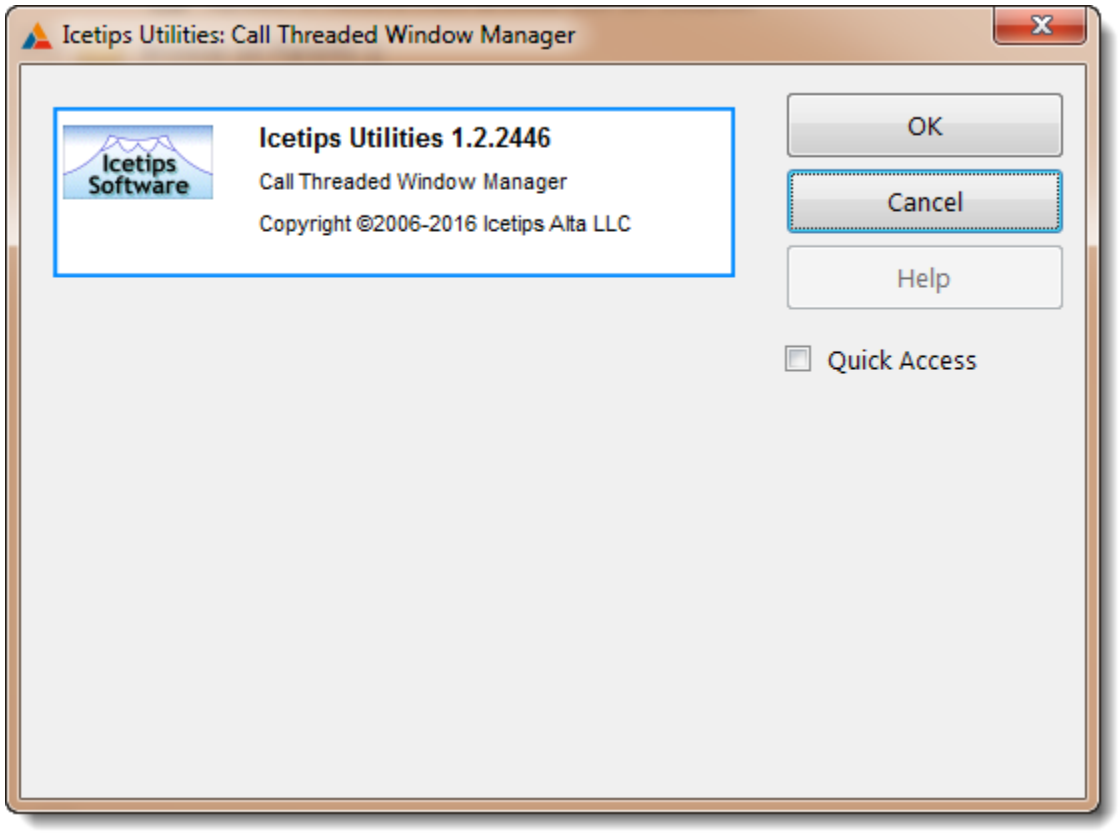


Column to change Select the listbox column to change. You can only select from columns that have the icon property set in the listbox formatter.

4.3.2.5 Icetips Call Threaded Window Manager

Extension Templates - Procedure Extensions

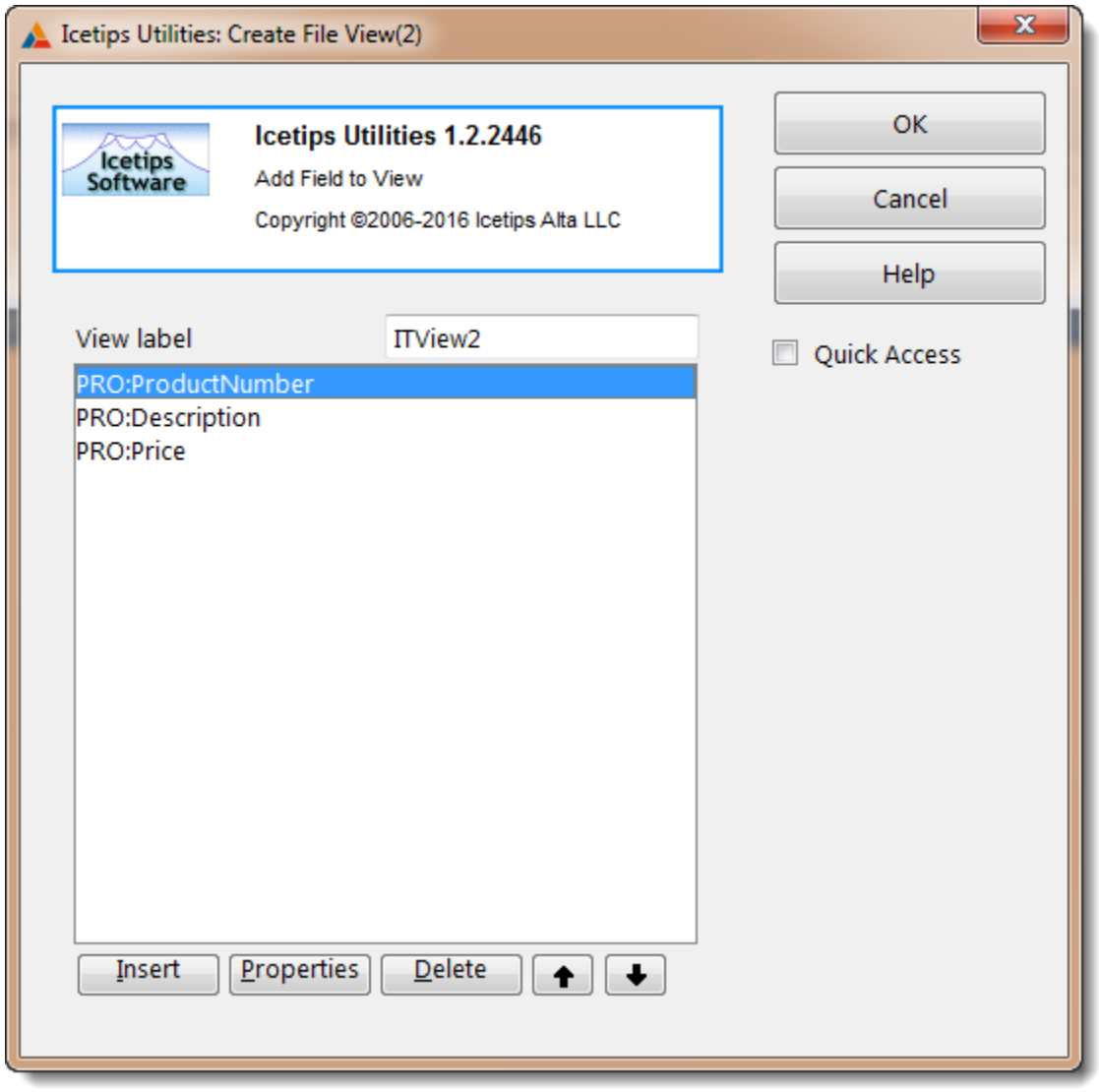
This procedure extension needs to be added to procedures that should be added when using the Global Threaded Window Manager 449 template.



4.3.2.6 Icetips Create File View

Extension Templates - Procedure Extensions

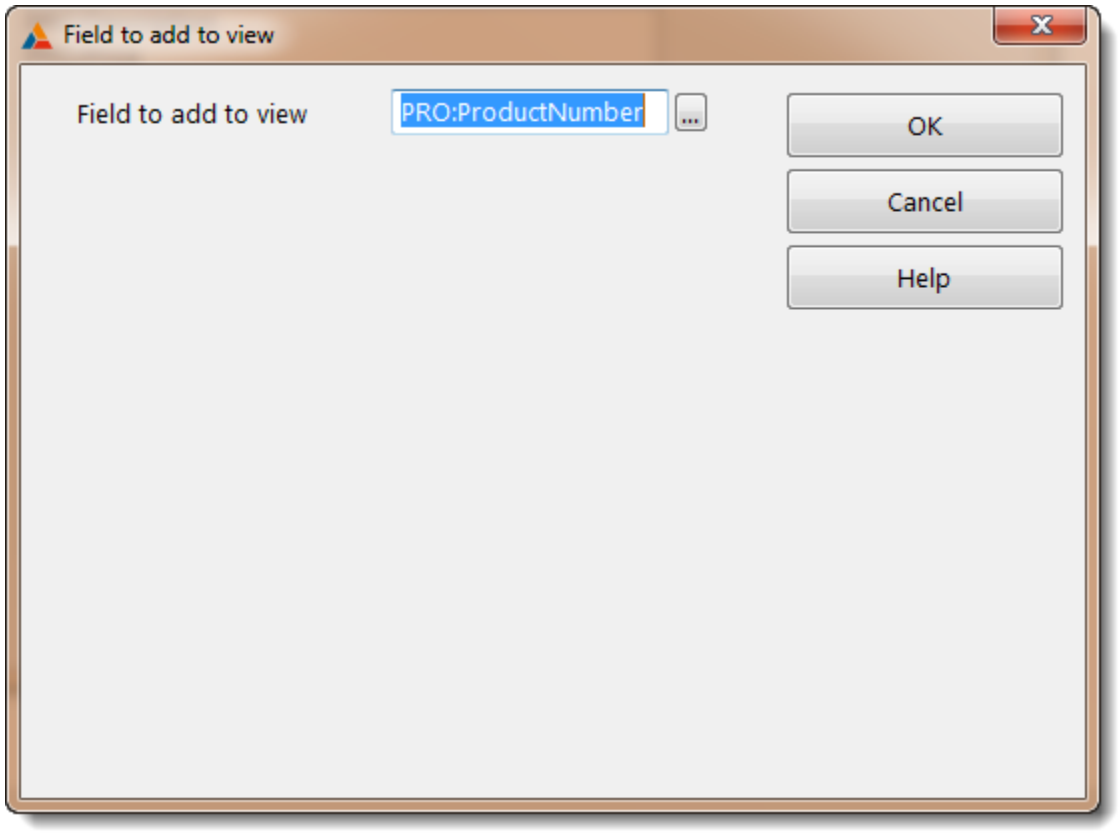
This templates allows you to create a regular VIEW structure. It can be useful when dealing with files, particularly SQL files where you only want certain columns from a table. It lets you set up a queue with corresponding fields to the ones in the view. This template is required by the "Fill Queue from SQL View" extension template, which loads the queue.



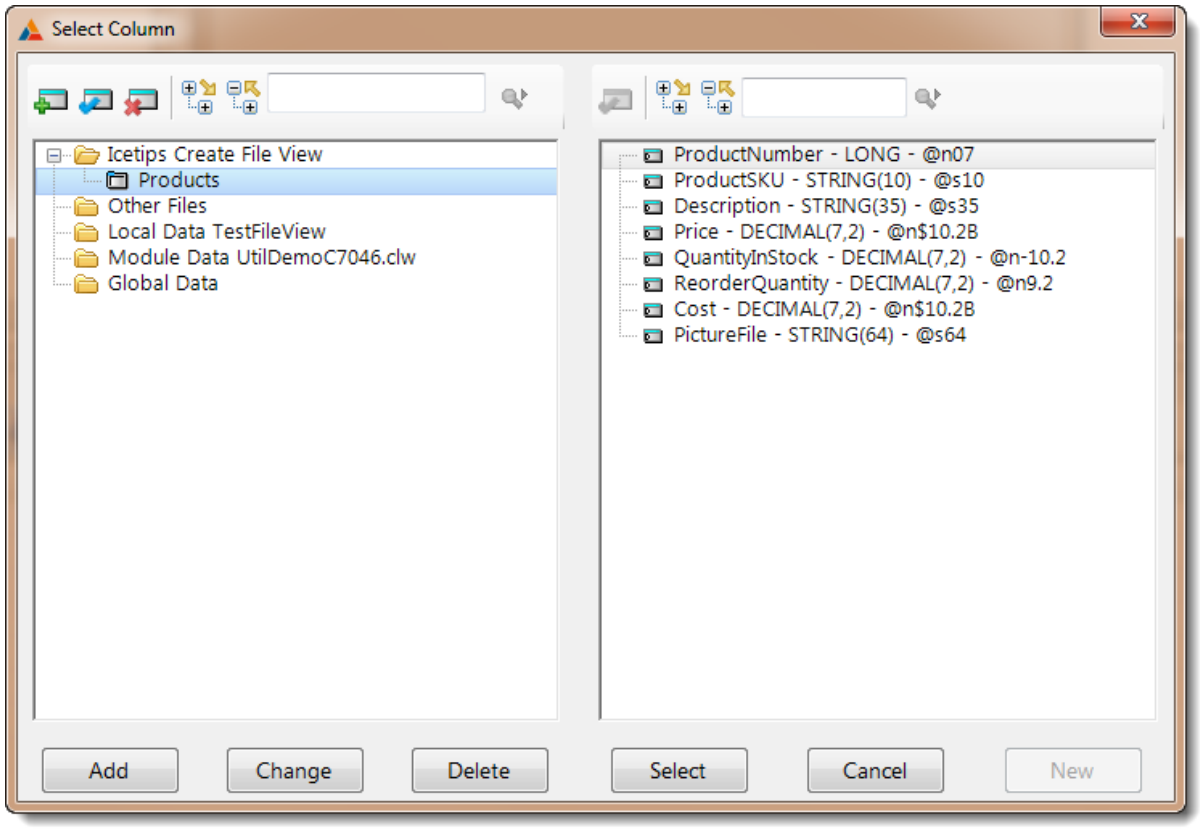
View label The label that is used to create the VIEW. It defaults to ITView and a number. You can change it to anything as long as it doesn't create a duplicate label.

Field list This is the list of fields that you select. The list above would generate the VIEW structure below:

```
ITView2                 VIEW(Products)
                          PROJECT(PRO:ProductNumber)
                          PROJECT(PRO:Description)
                          PROJECT(PRO:Price)
                        END
```



Field to add to view Select a field from the file schematic to add to the view as shown below.



4.3.2.7 Icetips Fill Queue from SQL View

Extension Templates - Procedure Extensions

This template requires the Create File View

4.3.2.8 Icetips Pre and post prime ABC Browse

Extension Templates - Procedure Extensions

Under construction.

4.3.2.9 Icetips Resize Options

Extension Templates - Procedure Extensions

Under construction.

4.3.2.10 Icetips Resize Options With Information

Extension Templates - Procedure Extensions

Under construction.

4.3.2.11 Icetips SQL Queue Process Construction

Under construction.

4.3.2.12 Icetips SQL Queue Report Construction

Under construction.

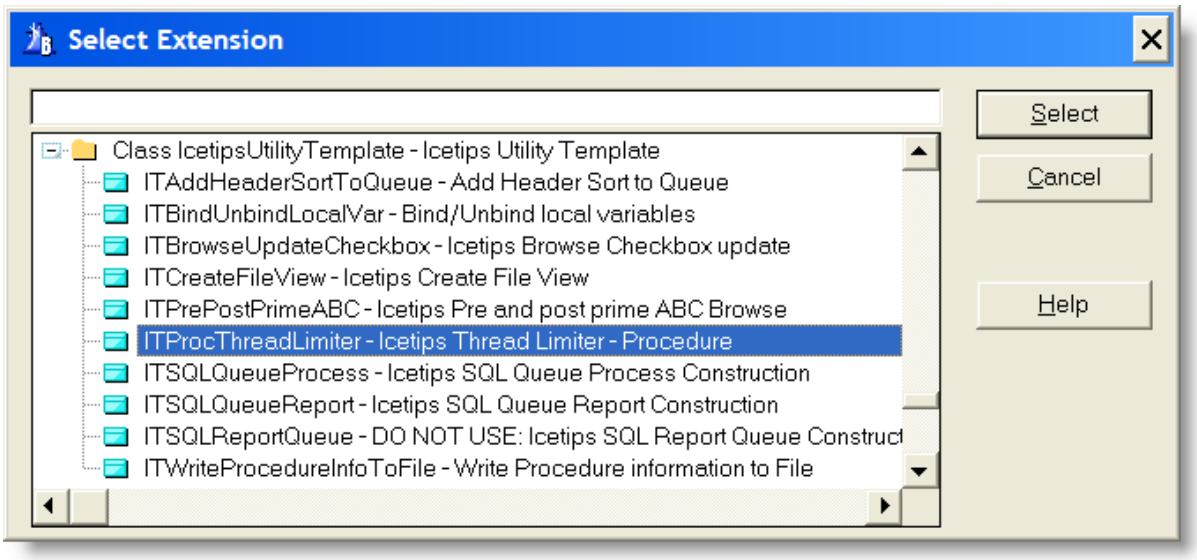
4.3.2.13 Icetips Thread Limiter - Procedure

This template enables thread limiting on a procedure. Select the "ITProcThreadLimiter - Icetips Thread Limiter - Procedure" template. Note that this template will only show up in the "Select Extension" window if the "Include Thread Limiter" is checked on the Icetips Utilities Classes Global 451 template.

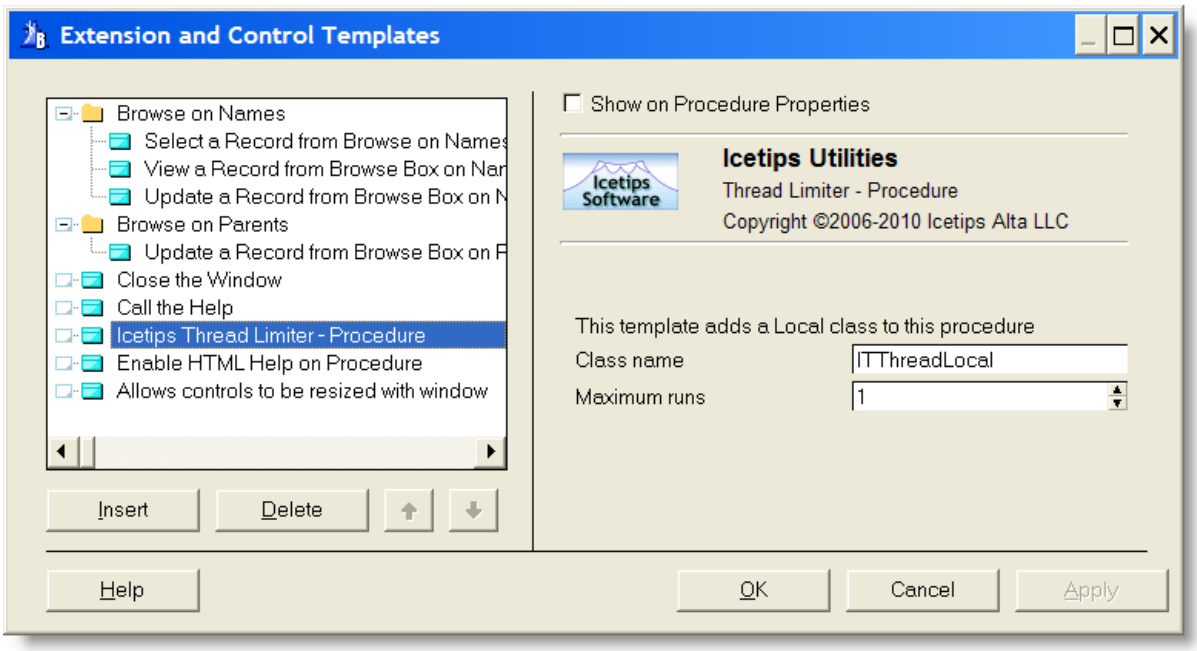
To use this template, check the "Include Thread Limiter" on the Icetips Utilities Classes Global 451 template. Then add the procedure extension (see below) and set up the "Maximum Runs" if necessary. Compile and run. For multi-DLL applications, you must check the "Include Thread Limiter" on both the DLL app that exports file declarations and also in the app where you need the Thread Limiter. **Make sure that you compile the exporting dll!**

The Thread Limiter keeps track of each time that a procedure with the procedure extension is opened. Note that the extension should work on all normal window procedures, but is not designed to work on reports or processes and has not been tested for that. If you need it to work on different type of procedures please let us know and we will add additional support for other procedure types.

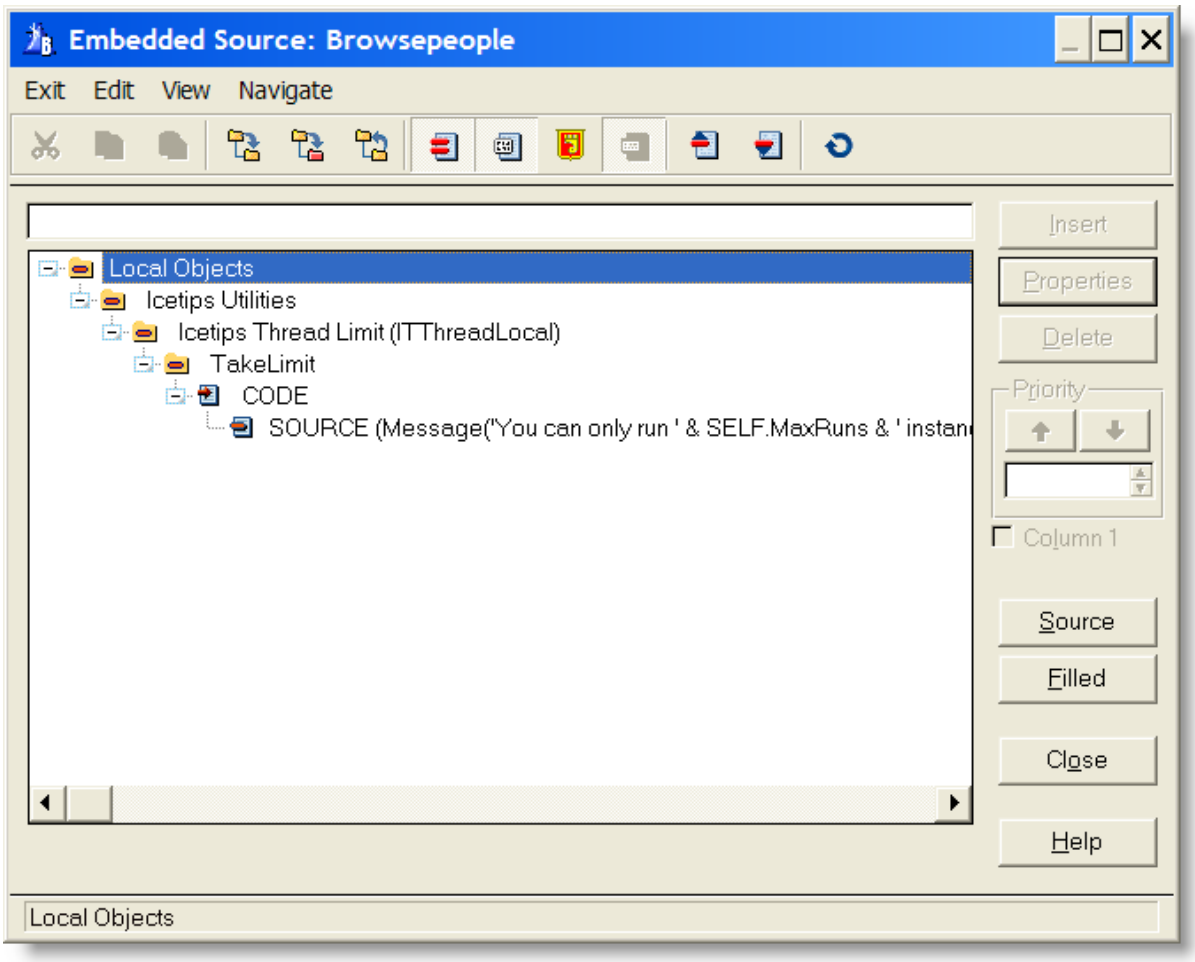
If it exceeds the maximum number of instances allowed, it will trigger the locally derived TakeLimit method to be called. By embedding code in that method you can control how you handle the limitation. The method calls its parent TakeLimit method which activates the last instance of the procedure. The activation brings the last instance to top, selects the window and if the window was minimized it is restored.



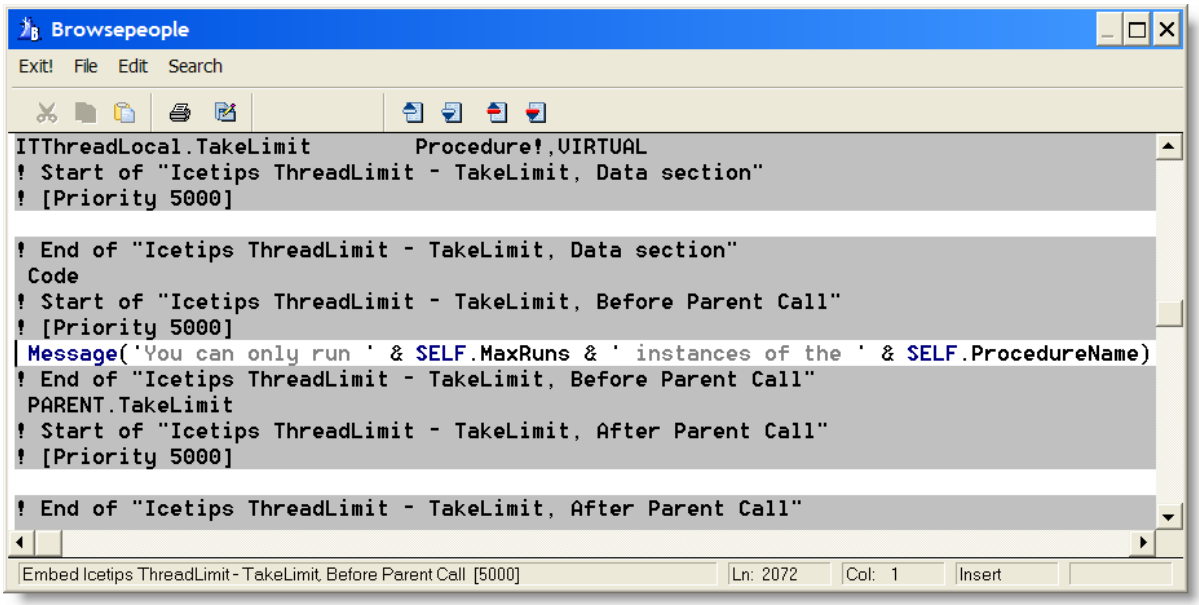
Once you have added it to the template you can change the classname and the maximum number of times the procedure will run.



The template adds embeds to override one method in the derived ITThreadLimitClass class, the TakeLimit method which is called when the procedure has already been run "Maximum runs" times. You can embed code into this embed for example to show a message or do some other activity.



You have an embed in the data section and two embeds in the code section, before and after parent call:

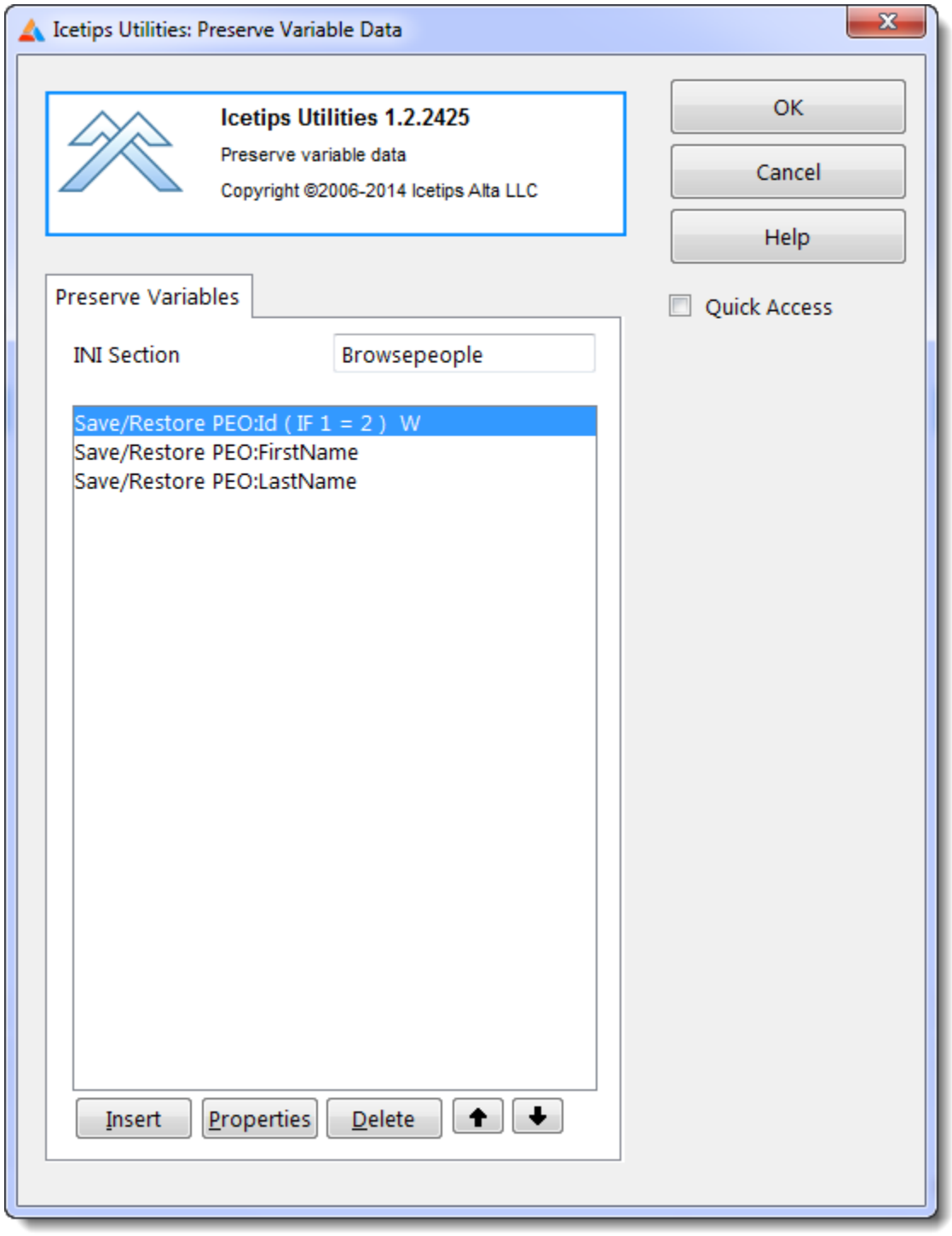


PARENT.TakeLimit will activate the last instance of the procedure, bring it into focus and bring it to top of other windows.

4.3.2.14 Icetips Preserve Variable Data

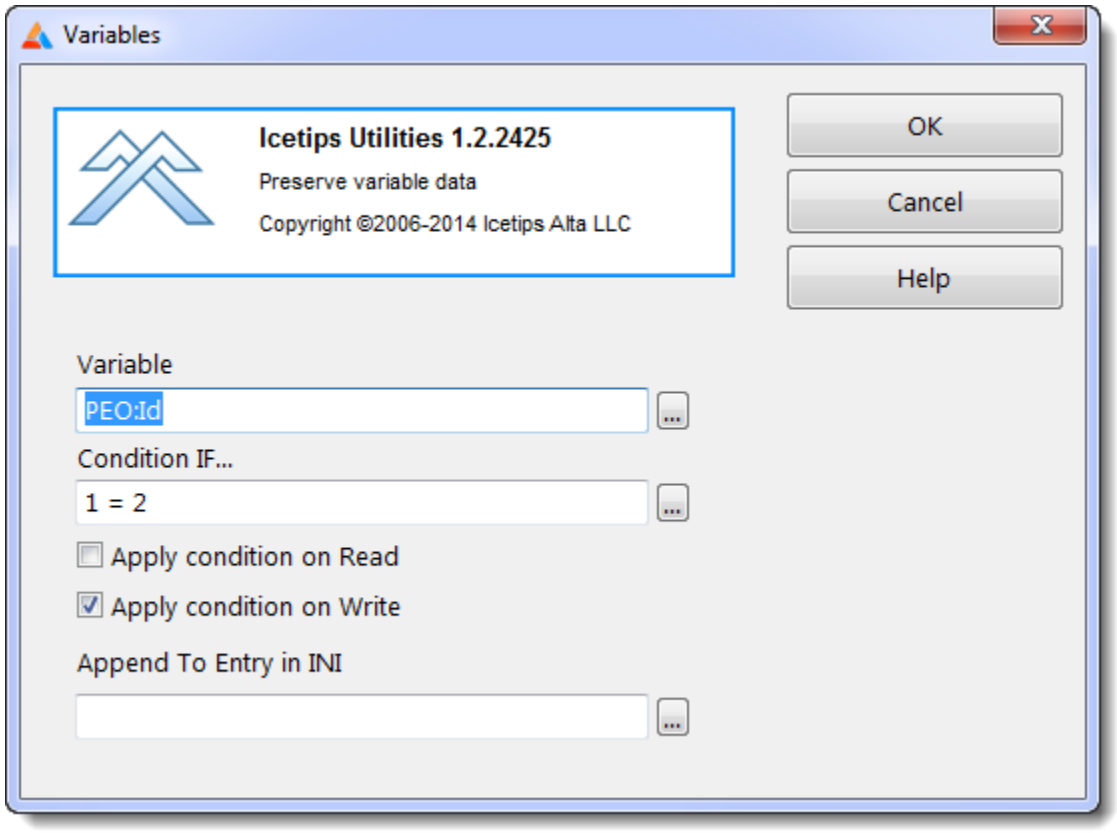
This template uses the INI Manager class to get and put data from the selected variables - can be application variables or database variables to set defaults for certain fields. Note that it only works at procedure level. The code executes as part of the WindowManager.Init and WindowManager.Kill to restore and save the values. It also works in Source procedures, where the read is put at priority 1 and the write is put at priority 9999. An optional condition can be specified for both restore and save. The default INI Manager class will store the data either in an INI file or in the Registry depending on the application settings you have.

By default the data is stored using the variable name in the INI section specified in the template window below. By using the "Append to Entry in INI" option for the variable, you can add some text to it, if there are chances that it might create a duplicate entry in the INI file or Registry.



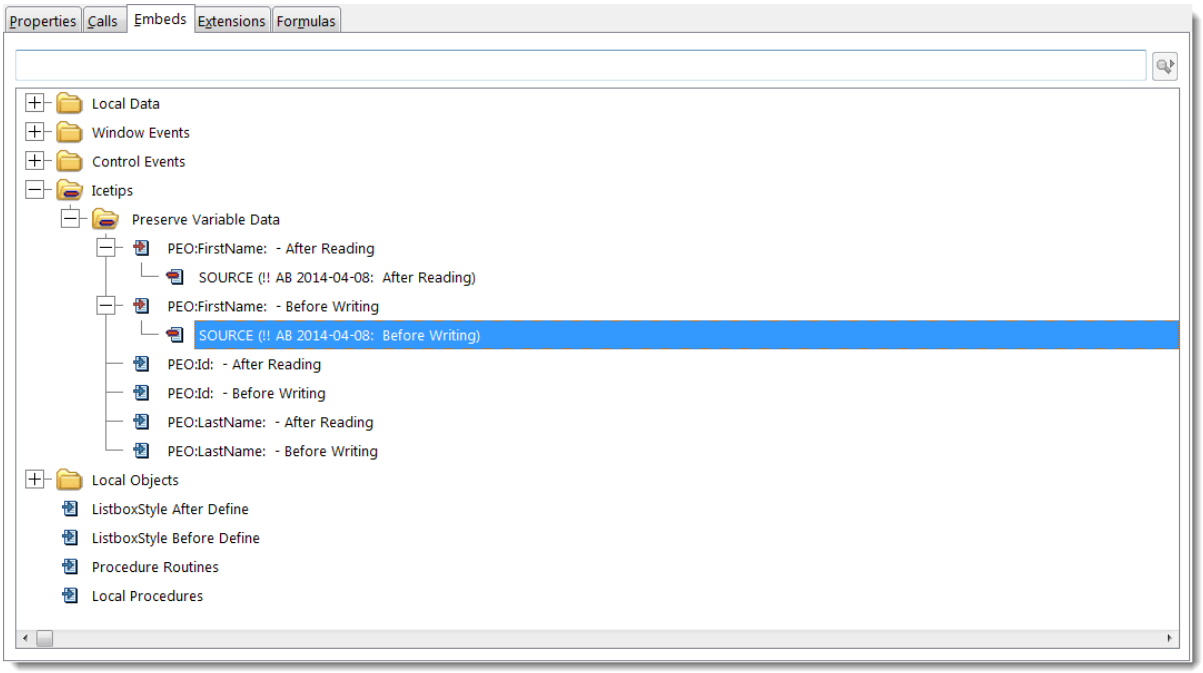
Simply add the variables you want to store.

INI Section The [SECTION] in the INI file or the KEY in the registry to put the value(s) in. This uses the Procedure name as default.



Variable	Select the variable you want to use. The variable must exist in the local data or in module or global data or from a data file.
Condition IF...	This allows you to set a condition expression that will be evaluated using an IF statement. You can select variables or type in a condition.
Apply condition on Read	Specifies if the condition should be applied when Reading the data.
Apply condition on Write	Specifies if the condition should be applied when Writing the data.
Append To Entry in INI	This is an optional string that you can enter that will be appended to the entry in the INI file. By

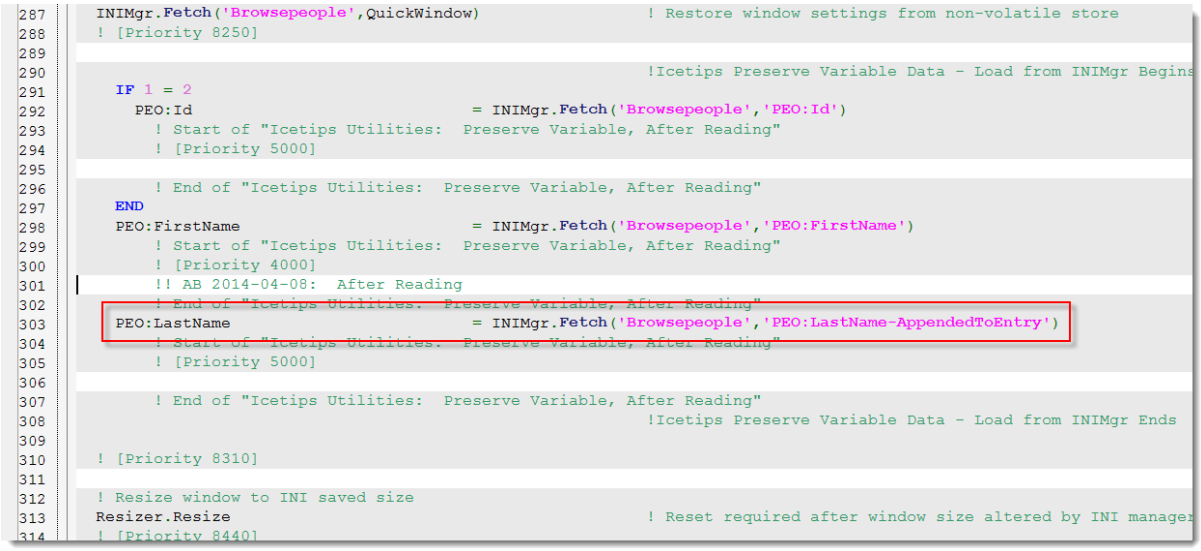
Each variable has it's own embeds, before and after it is assigned, where you can possibly prime the variable with a value right before it is saved, or do something with it after it is read.



This results in this generated code:

```
288   ! [Priority 8250]
289
290                                                    !Icetips Preserve Variable Data - Load from INIMgr Begins
291     IF 1 = 2
292       PEO:Id                              = INIMgr.Fetch('Browsepeople','PEO:Id')
293         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
294         ! [Priority 5000]
295
296         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
297     END
298     PEO:FirstName                           = INIMgr.Fetch('Browsepeople','PEO:FirstName')
299         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
300         ! [Priority 4000]
301         !! AB 2014-04-08:  After Reading
302         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
303     PEO:LastName                            = INIMgr.Fetch('Browsepeople','PEO:LastName')
304         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
305         ! [Priority 5000]
306
307         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
308                                                    !Icetips Preserve Variable Data - Load from INIMgr Ends
309
310   ! [Priority 8310]
311
312   ! Resize window to INI saved size
313   Resizer.Resize                                   ! Reset required after window size altered by INI manager
314   ! [Priority 8440]
315
```

If the PEO:LastName had "-AppendToEntry" entered in the "Append To Entry in INI" field, the generated code would look like this:



```
287   INIMgr.Fetch('Browsepeople',QuickWindow)                ! Restore window settings from non-volatile store
288   ! [Priority 8250]
289
290                                                           !Icetips Preserve Variable Data - Load from INIMgr Begins
291     IF 1 = 2
292       PEO:Id                              = INIMgr.Fetch('Browsepeople','PEO:Id')
293         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
294         ! [Priority 5000]
295
296         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
297     END
298     PEO:FirstName                       = INIMgr.Fetch('Browsepeople','PEO:FirstName')
299         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
300         ! [Priority 4000]
301         !! AB 2014-04-08:  After Reading
302         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
303     PEO:LastName                        = INIMgr.Fetch('Browsepeople','PEO:LastName-AppendedToEntry')
304         ! Start of "Icetips Utilities:  Preserve Variable, After Reading"
305         ! [Priority 5000]
306
307         ! End of "Icetips Utilities:  Preserve Variable, After Reading"
308                                                           !Icetips Preserve Variable Data - Load from INIMgr Ends
309
310   ! [Priority 8310]
311
312   ! Resize window to INI saved size
313   Resizer.Resize                                          ! Reset required after window size altered by INI manager
314   ! [Priority 8440]
```

4.3.2.15 Write Procedure information to File

Enter topic text here.

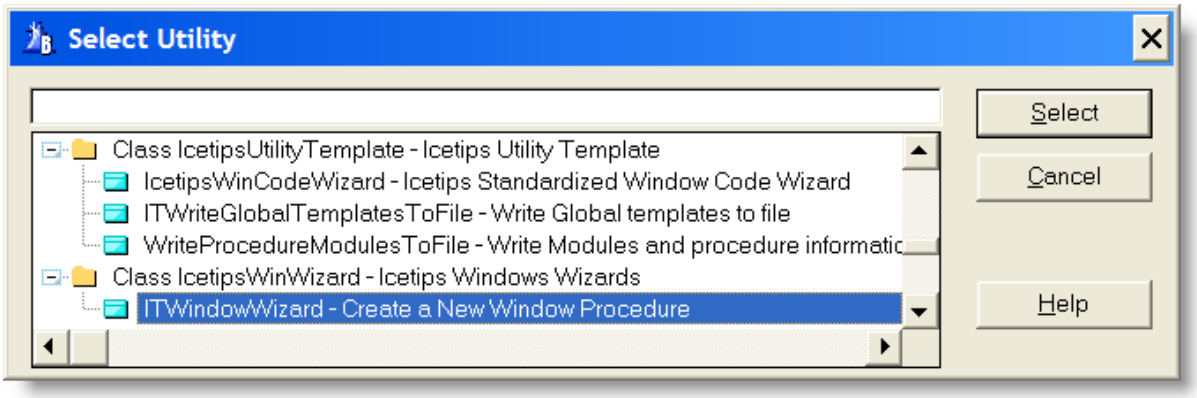
4.4 Utility Templates

The Icetips Utilities currently contain 6 utility templates:

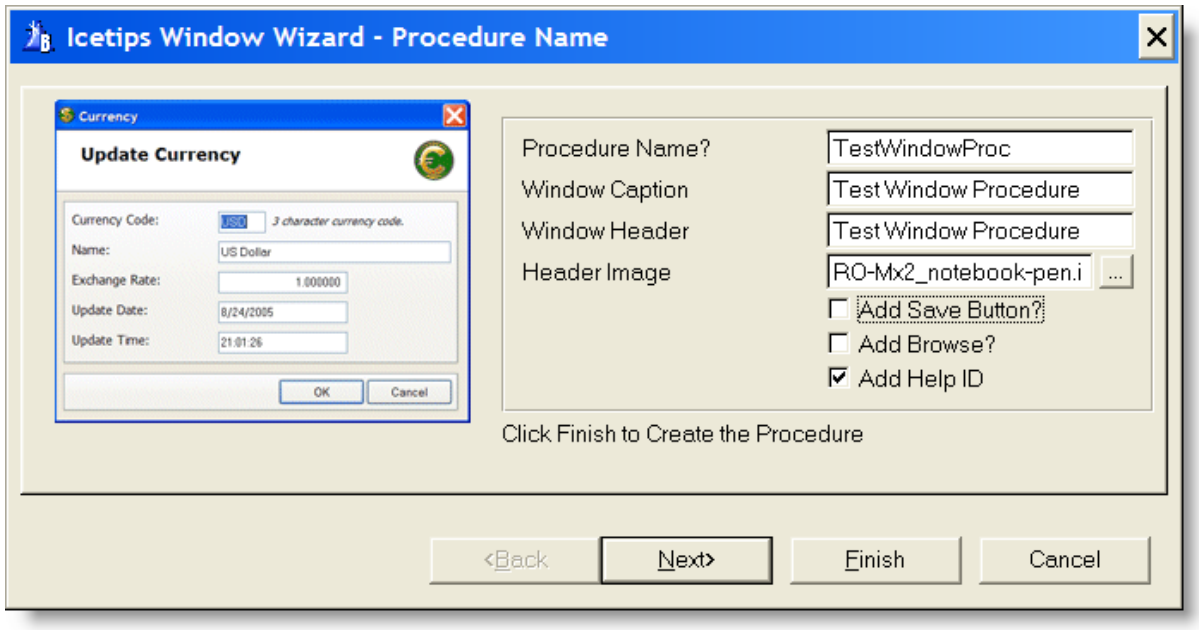
4.4.1 Create a New Window Procedure Utility Templates

This wizard template creates a new window procedure. Due to problems with the #IMPORT template statement in Clarion, we have had to resolve to letting the wizard create a TXA file that you have to import manually. Using the #IMPORT template statement could result in frequent crashes of the Clarion IDE when using the wizard template.

To start the wizard, either press Ctrl-U on the keyboard or select "Application | Template Utility..." from the Clarion IDE main menu. That will bring up the "Select Utility" window. Locate the "IcetipsWinWizard - Icetips Windows Wizard" class and then "ITWindowWizard - Create a New Window Procedure" - see the screenshot below.

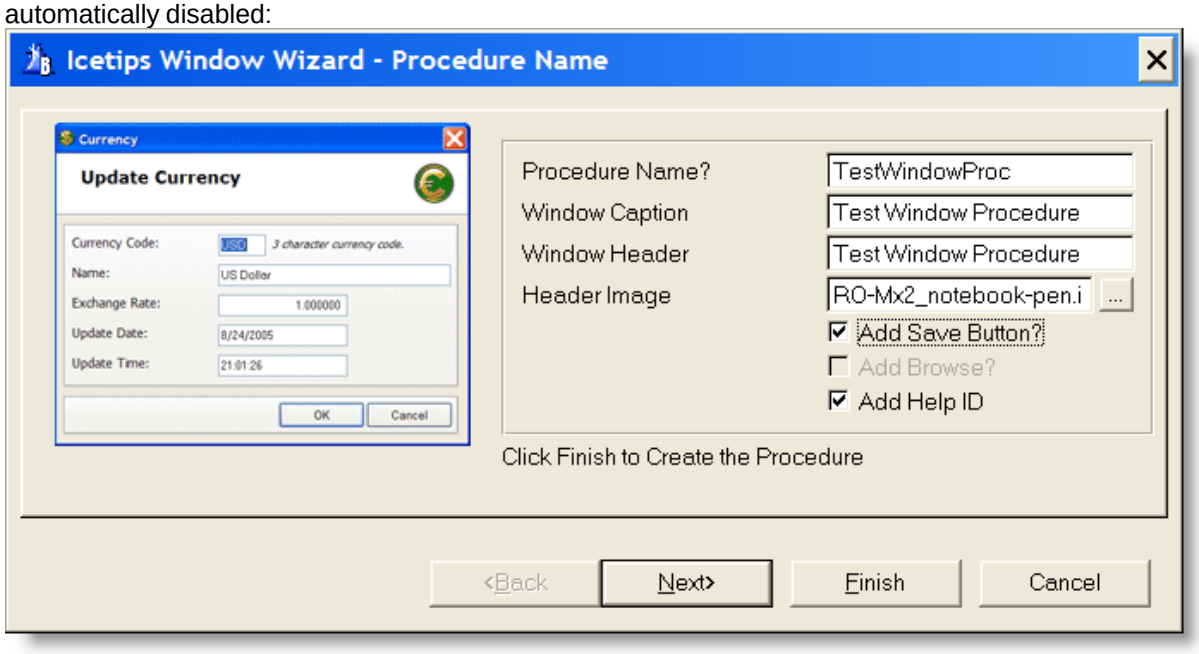


The first screen prompts for procedure name and also the window caption and header text. When the caption is entered it is automatically copied to the Window Header. You can choose to create a plain window, a window with a save button (form) or a window with a browse.

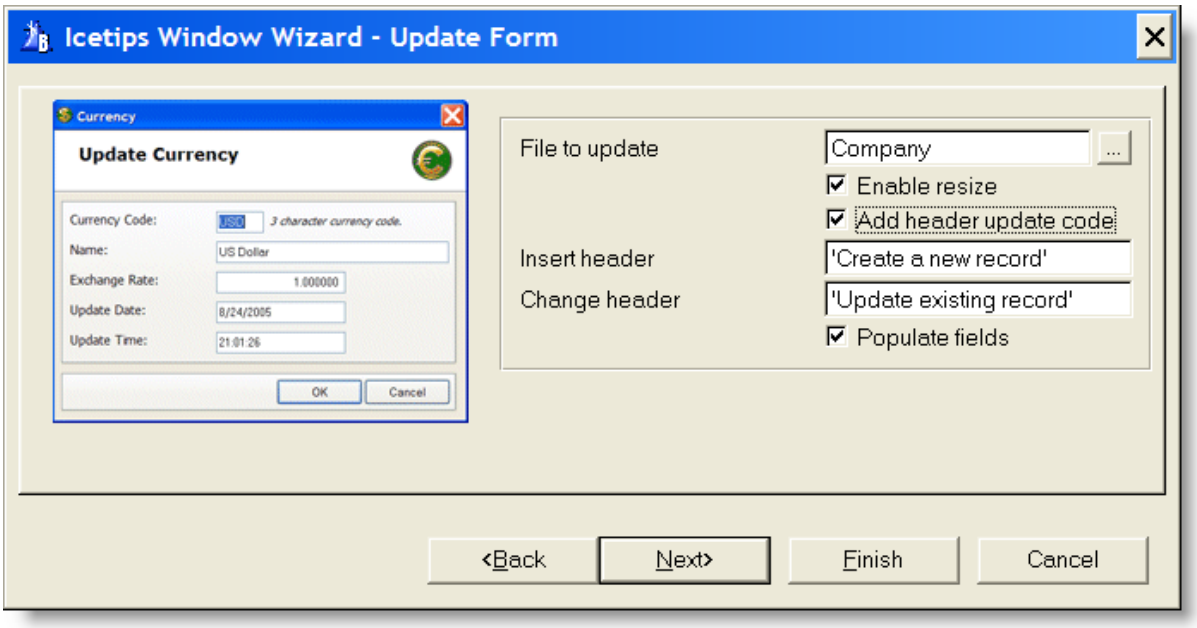


For a plain window, leave the "Add Save Button?" and "Add Browse?" checks unchecked. "Add Help ID" will add the procedure name as a help id for the window, i.e. it adds HELP('TestWindowProc') if that is the procedure name you entered. "Add Help ID" is checked by default.

For a form window, check the "Add Save Button?" When you do that, the "Add Browse?" check is automatically disabled:



The next window prompts you for what file to use for the update form.

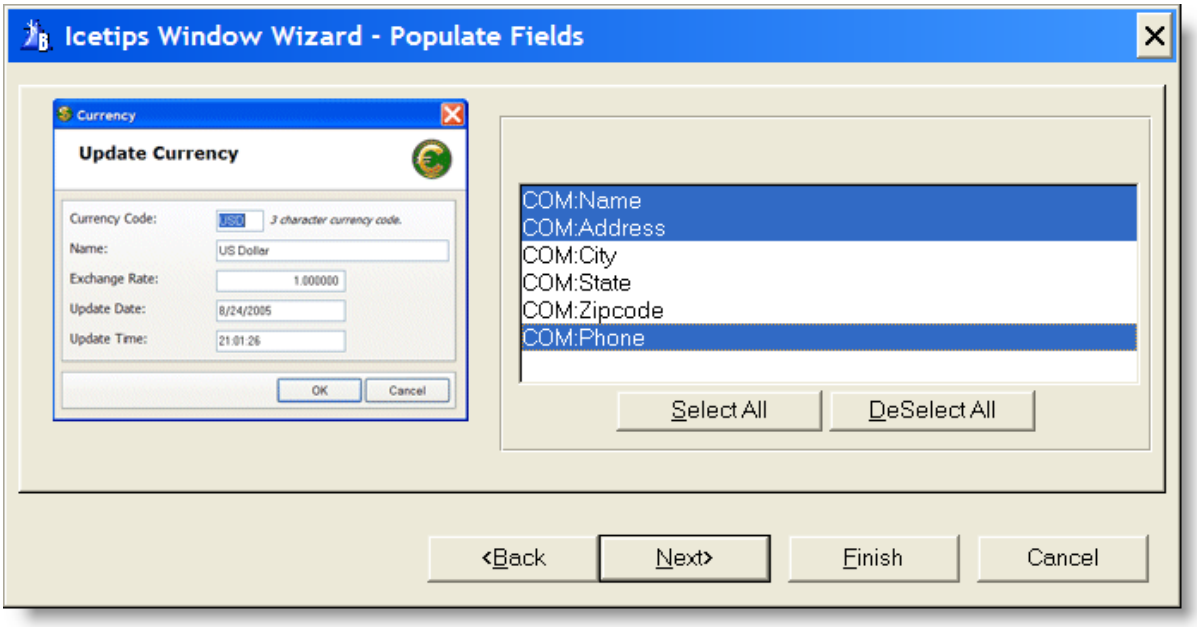


Enable Resize is not checked by default. If it is added resize template is added to the window. Note that the wizard does not attempt to set up the resizing, just adds the template to the window.

Add header update code is not checked by default. It adds code to show the strings in the "Insert Header" and "Change Header" in the header panel when inserting or changing record.

Populate fields enables the next window in the wizard where you can select fields to populate on the form. Note that no tabs are created on the window. The form will extend down to contain upto a maximum of 10 fields. Currently this maximum is fixed in the template code, but we will make it adjustable in future releases.

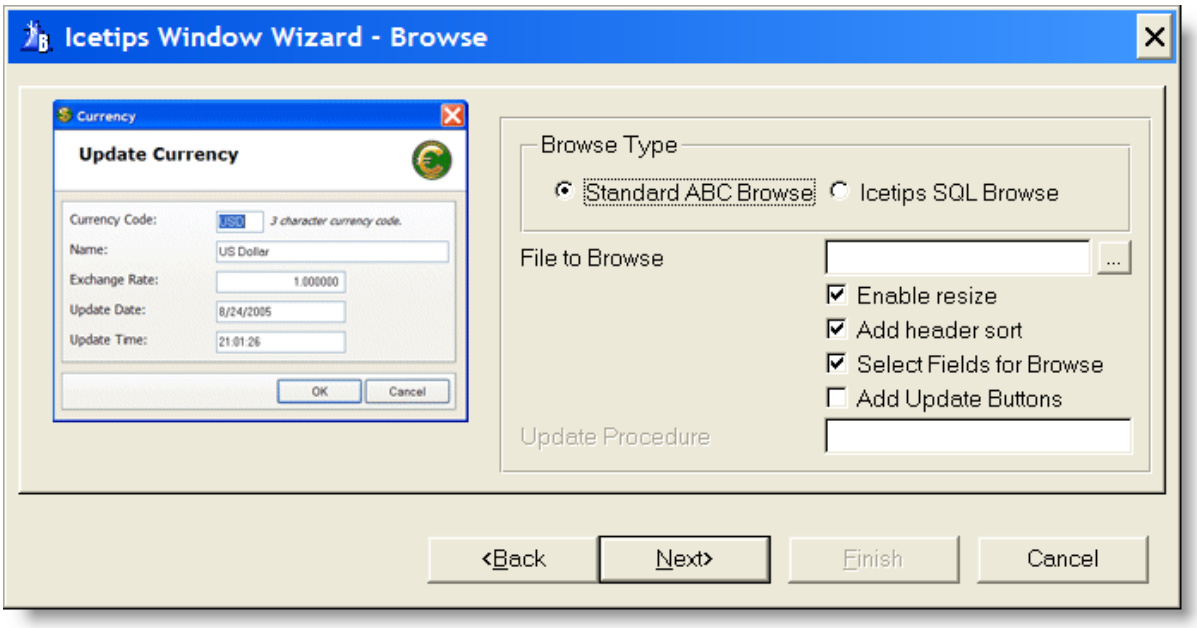
The next window allows you to pick what fields to add to the form window.



Simply click on the fields you want to add. You can also use the "Select All" and "DeSelect All" buttons

to do just that - select all the fields or deselect them all.

If you opted to create a browse instead of a form, you will get a slightly different sequence after the first window.



The first option is to select between a standard ABC browse or the Icetips SQL browse. The only difference is that when the Icetips SQL browse is selected, the "Add header sort" is disabled since the Icetips SQL browse always uses header sort.

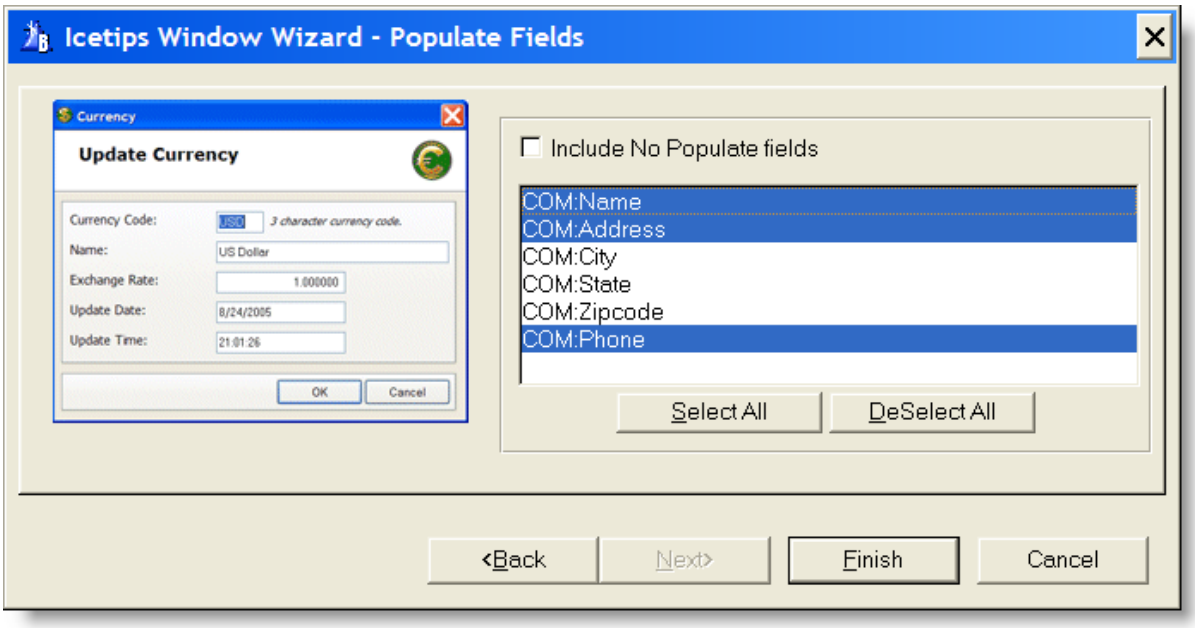
You are prompted to choose the file to use in the browse. Note that only a single file is selected and parent/child files cannot added to the browse at this point.

Enable resize is checked by default and adds the resize template to the window. Again, same as with the forms, this only means that the resize template is added and no code is generated to attempt to set resize options for individual controls.

Add header sort (ABC browses only) adds the header sort option to the browse. This option is not enabled for the Icetips SQL since it always uses header sorting.

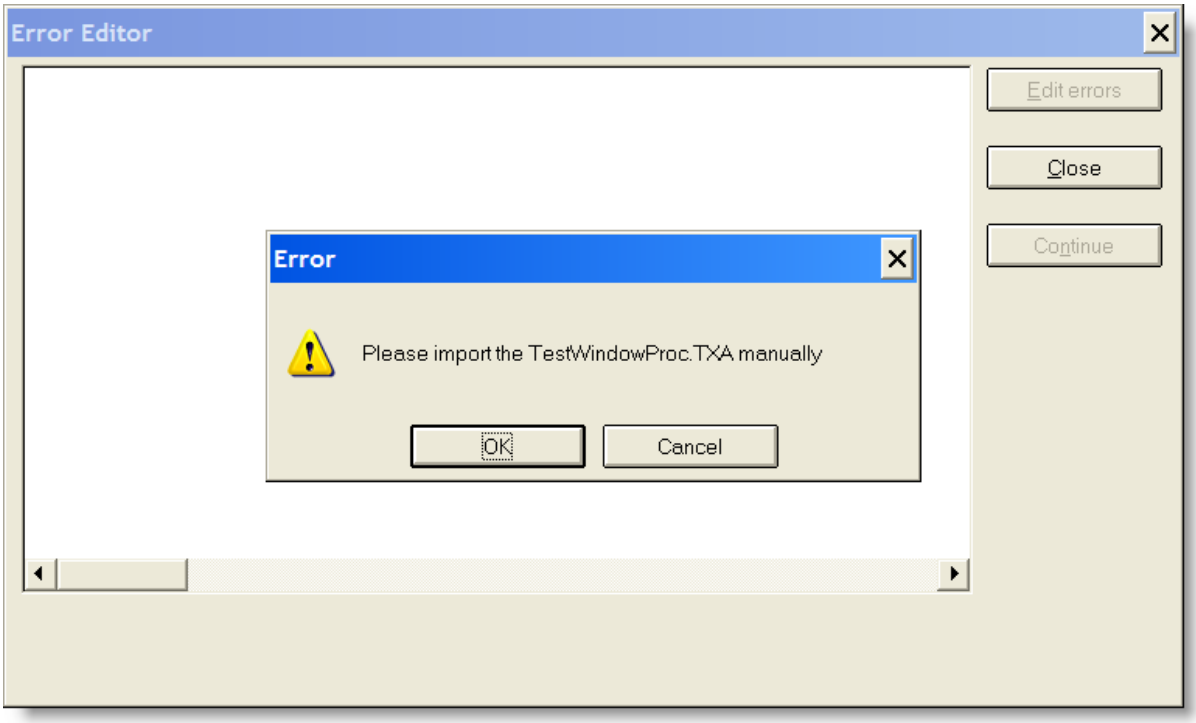
You can optionally select what fields to choose for the browse, this is done in the next screen. If you want to add the fields later, just uncheck this option.

Add Update Buttons is not checked by default. You can add the update buttons and then optionally specify the update procedure. Note that adding the update buttons does not automatically create a form - this wizard only creates one procedure at a time.

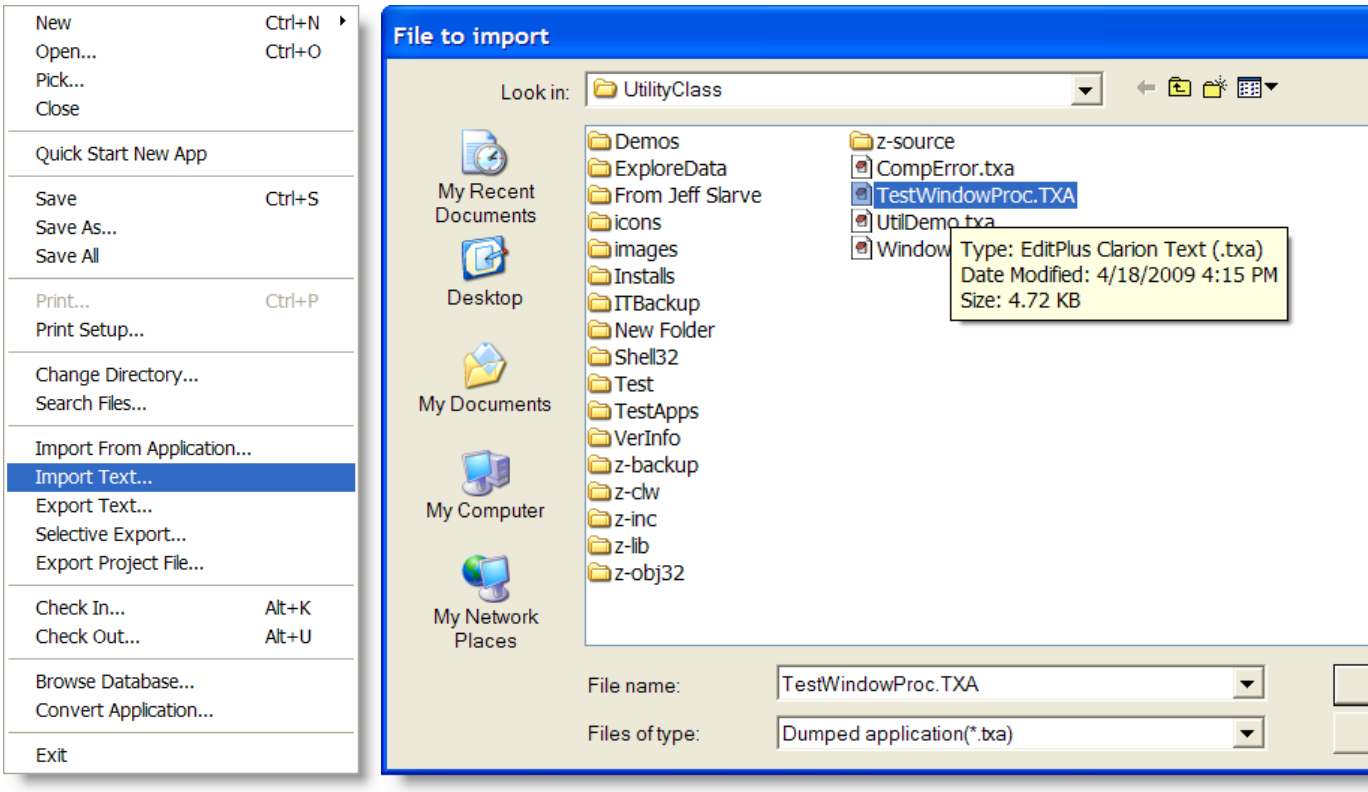


On this window you can select what fields to add to the browse. The "Include No Popluate fields" checkbox is cleared by default. When checked it will refresh the field list to show any fields that have the "No Populate" option checked in the dictionary. In some cases you may want to show those fields on browses.

When you click the "Finish" button you will be faced with an error window!



This tells you all that you need to do. The wizard created a TXA file in the application folder with the name that you entered for the procedure. Because of the problem with the #IMPORT statement as described above, you need to manually import the TXA. To do that, select "File | Import Text..." from the Clarion IDE main menu.

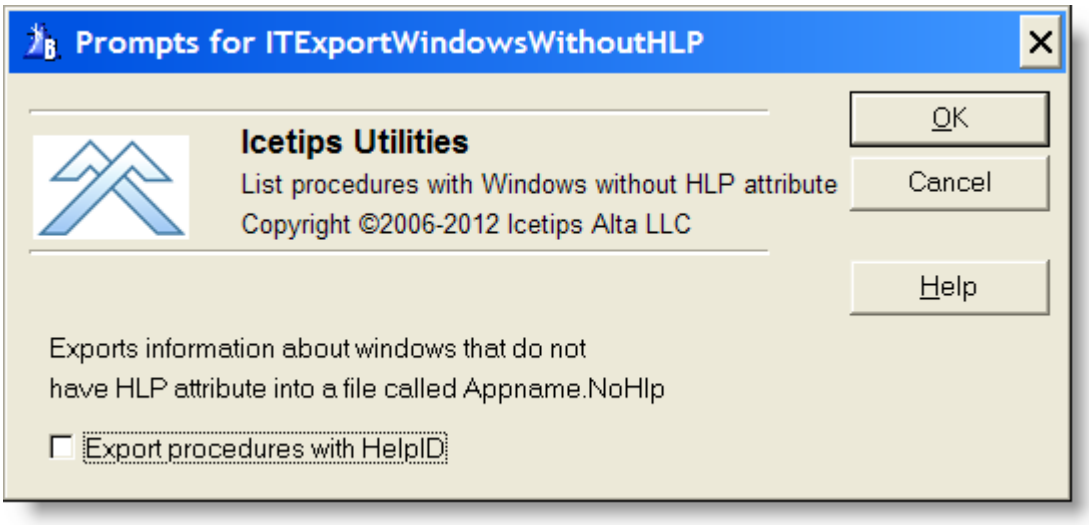


Then pick the file from the "File to Import" dialog and the Clarion IDE will import it to the application. Note that if you created a browse window and added the update buttons you will get a ToDo procedure for the update form. Double click on it and run this wizard again to create the form.

4.4.2 Export Windows without Help ID Utility Templates

This template exports a list of procedures that have windows, but no HLP attribute, i.e. no help ID. It exports to a file with the same name as the application with a ".NoHlp" extension. I.e. if your application is myapp.app, then the template will export to myapp.NoHlp.

This is very useful to keep track of windows that haven't been documented or haven't had a Help ID assigned to them.



The .NoHlp file is a simple list of procedures that have no help ID. If the **Export procedures with HelpID** is checked

```
BrowseDemo
BrowseDemoForm
ColorDemo                       'ColorDemoWindow'
ControlTypesDemo
IExplorerDemo
IconsDemo
Main                            'Appframe_Main'
MenuTest
NotepadDemo
OutlookDemo
StylesDemo
TabbedViewDemo
Welcome
```

If **Export procedures with HelpID** is not checked, the list would look like this:

```
BrowseDemo
BrowseDemoForm
ControlTypesDemo
IExplorerDemo
IconsDemo
MenuTest
NotepadDemo
OutlookDemo
StylesDemo
TabbedViewDemo
Welcome
```

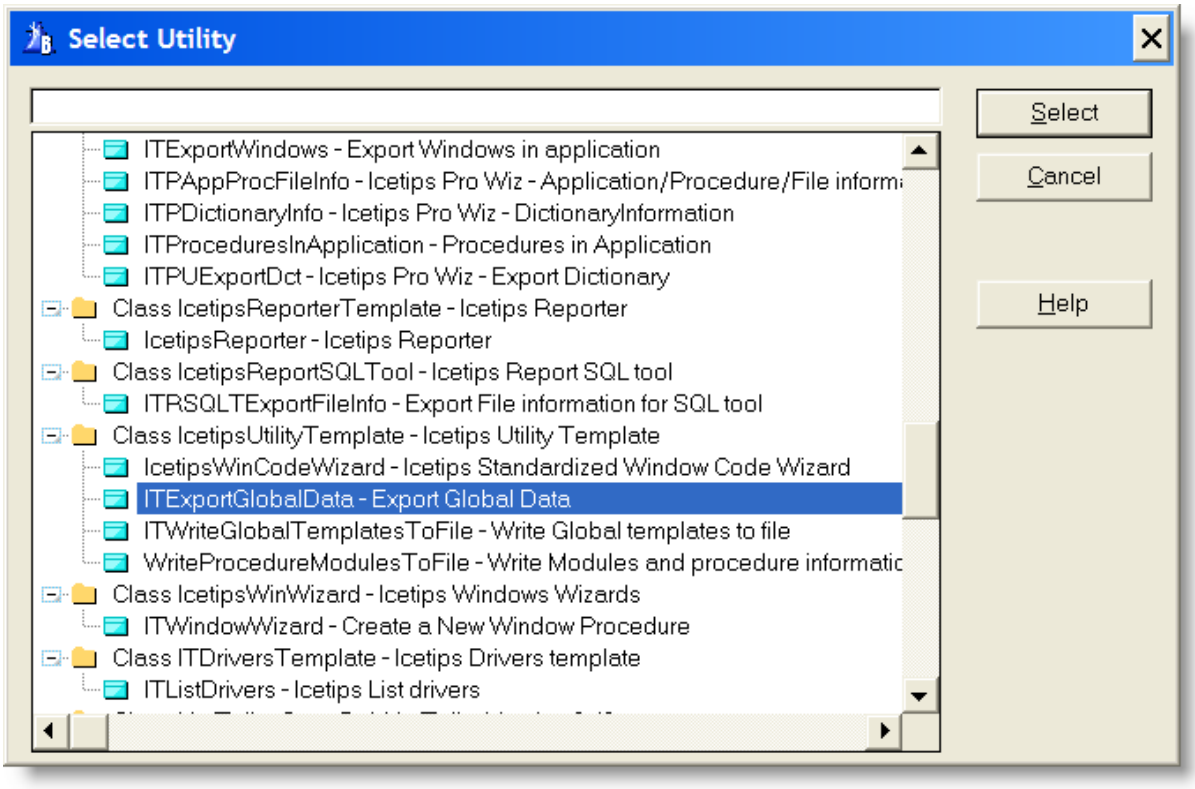
4.4.3 Export Global Data — Utility Templates

Note: This template no longer exits. Please use the "Prepare Multi-DLL app 496" template to export global data.

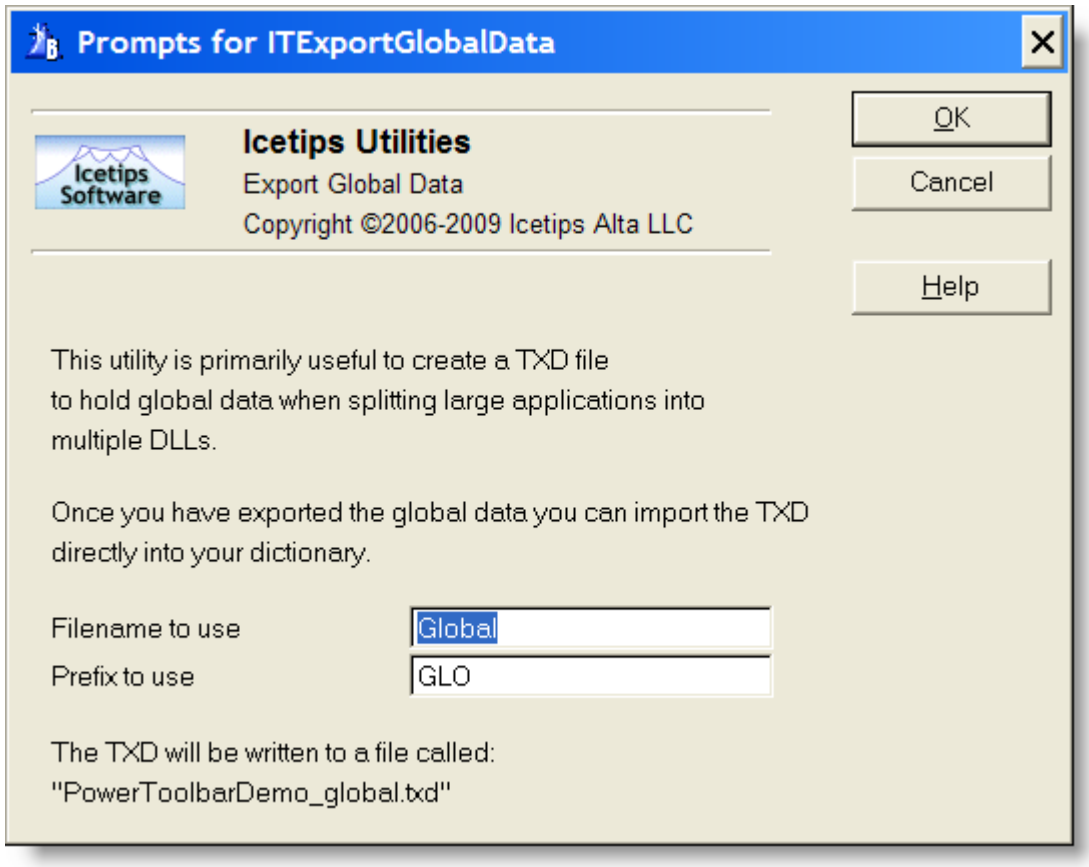
This template exports all global data that is in the application to a TXD file. You can then simply import

the TXD file into your dictionary to get all your global data from the application. This is very handy when splitting a single application file into multiple DLLs. Then it is very convenient to keep the Global data in the dictionary and then the templates will take care of exporting them correctly.

To start, select "Application | Template Utility" from the Clarion 6 main menu or "Application | Utility Template" from the Clarion 7 menu. Alternatively you can hit Ctrl-U on the keyboard (both versions)

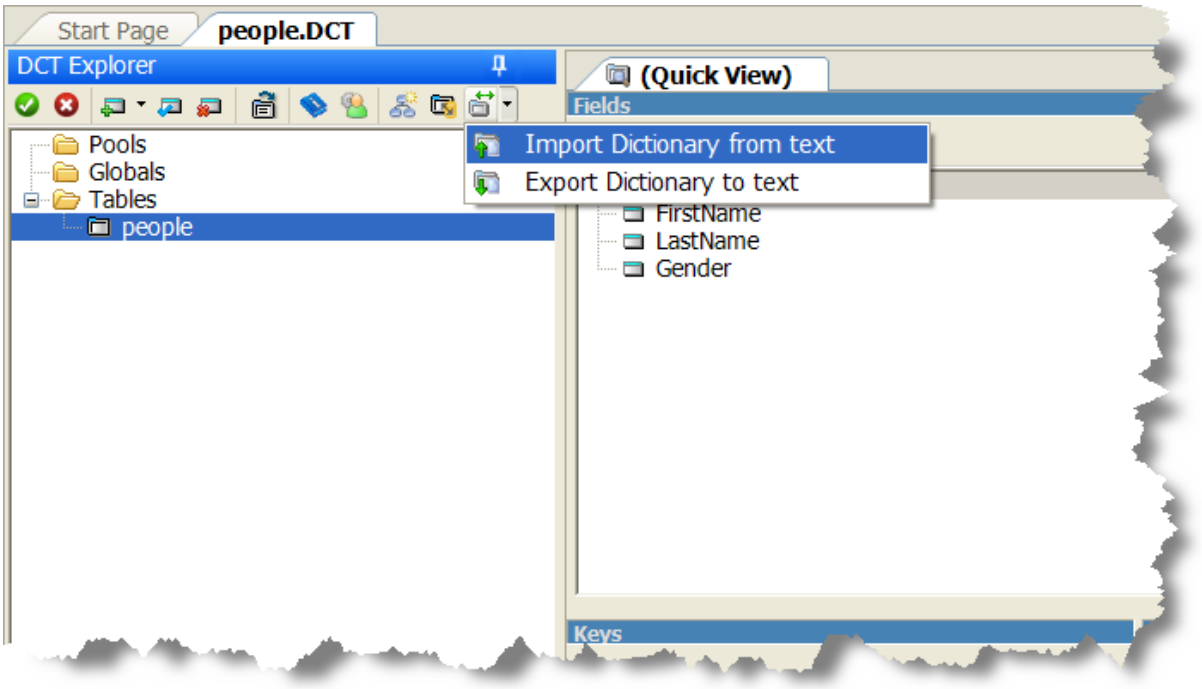


Select the "ITExportGlobalData - Export Global Data" utility template from the "IcetipsUtilityTemplate" class.



By default the filename to use is set to "Global" and the prefix to "Glo" If this does not match with what you need you can change it before you export the data. Click the OK button to export the data. It will be exported to a file that uses the same filename as the application (.app file) but with a "_global" appended to the name. This is a fully qualified TXD file so it can be imported directly, however only the label and data type are exported.

The next step is import the .txd file into your dictionary. Load the target dictionary file and select "File | Import Text" in Clarion 6. In Clarion 7 please see the screenshot below:



Example:

```
[DICTIONARY]
VERSION '1.0'
CREATED '27 SEP 2009' ' 5:49PM'
MODIFIED '27 SEP 2009' ' 5:49PM'

[FILES]
Global FILE,DRIVER('TOPSPEED'),PRE(GLO),CREATE,THREAD
!!> USAGE(Global)
Record                    RECORD
BrowseInfoShown             BYTE
Test                        STRING(20)
                          END
                        END
```

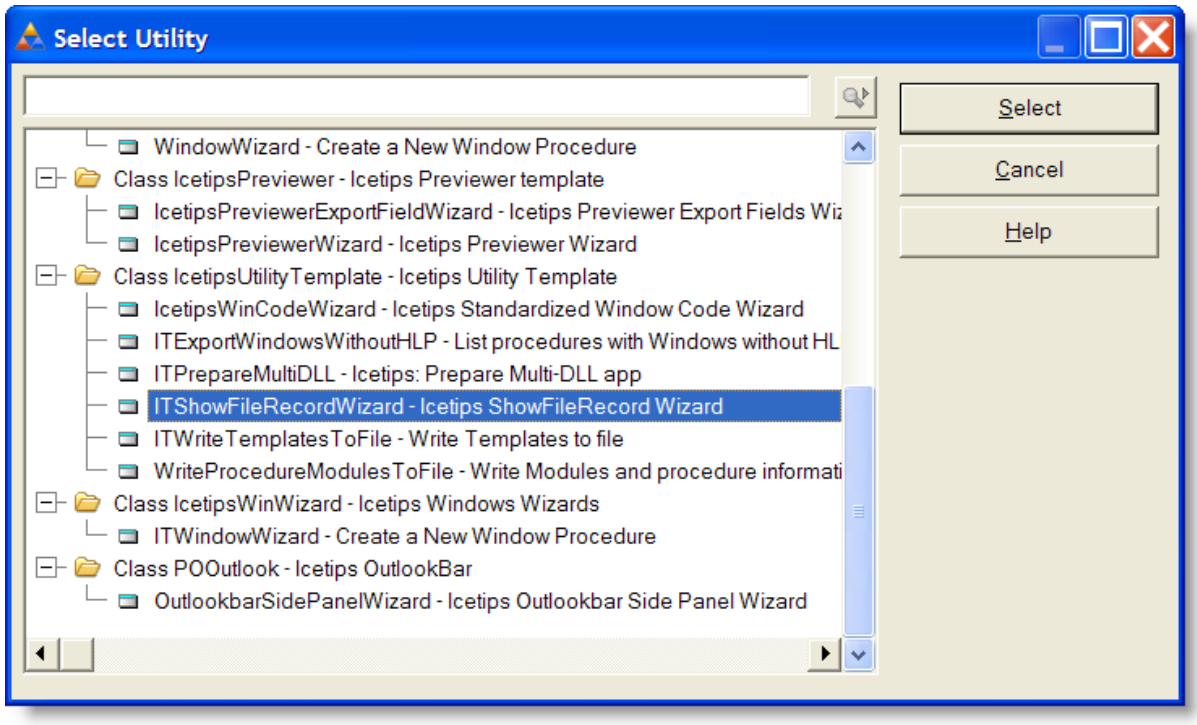
This shows a generated TXD with two Global fields in it, ready to import into a dictionary.

4.4.4 Icetips Create ShowFileRecord Wizard Utility Templates

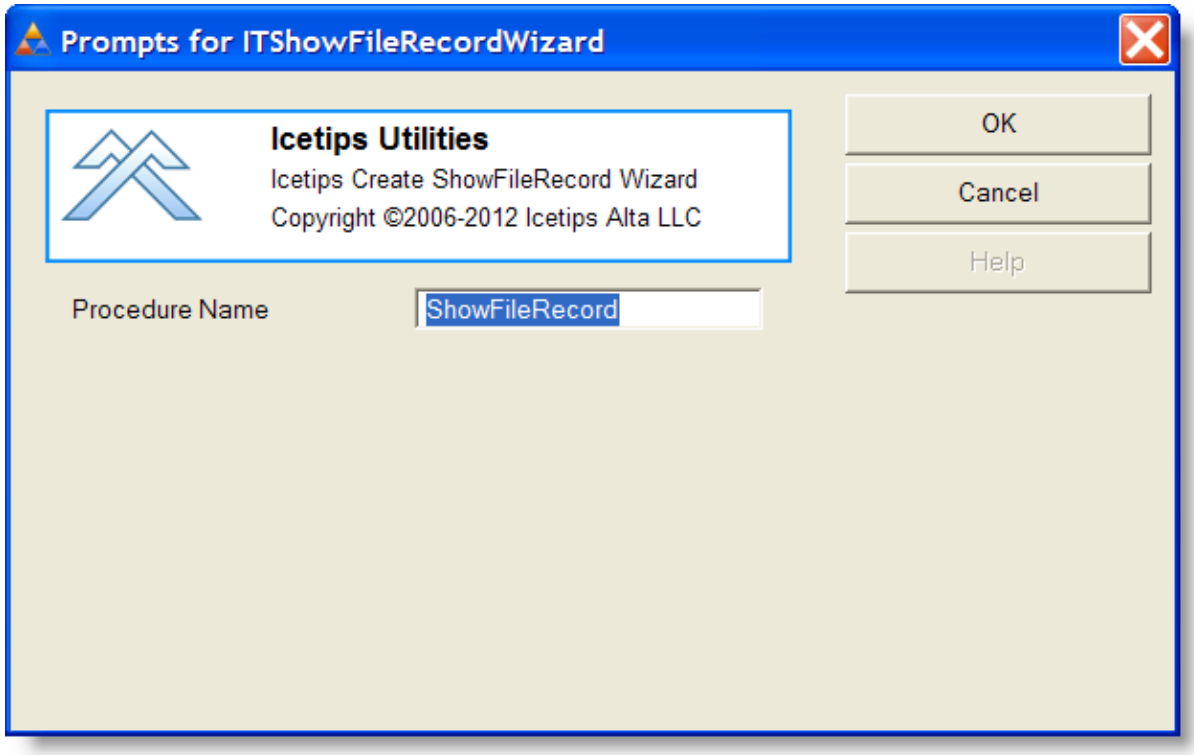
The ShowFileRecord procedure is a very useful tool to debug data in browses. It retrieves the structure and data from the primary file in the browse and shows you the data in each column. It also shows you information about the table structure as well as the data types for each column so it is easy to see what is going on. Since it reveals information about the file properties and the columns, it can be a quick way to view those properties at runtime, particularly if access to the dictionary is not possible, for example when working on site and there is no access to the code from there.

To create the ShowFileRecord use this utility template. You can then use the Global Call ShowRecord from Browse 439 global extension template to quickly and easily add this procedure to all browses in your application.

To create the ShowFileRecord procedure open the "Select utility" window, either by using "Application | Utility Template" from the Clarion IDE main menu or by pressing Ctrl-U on the keyboard.

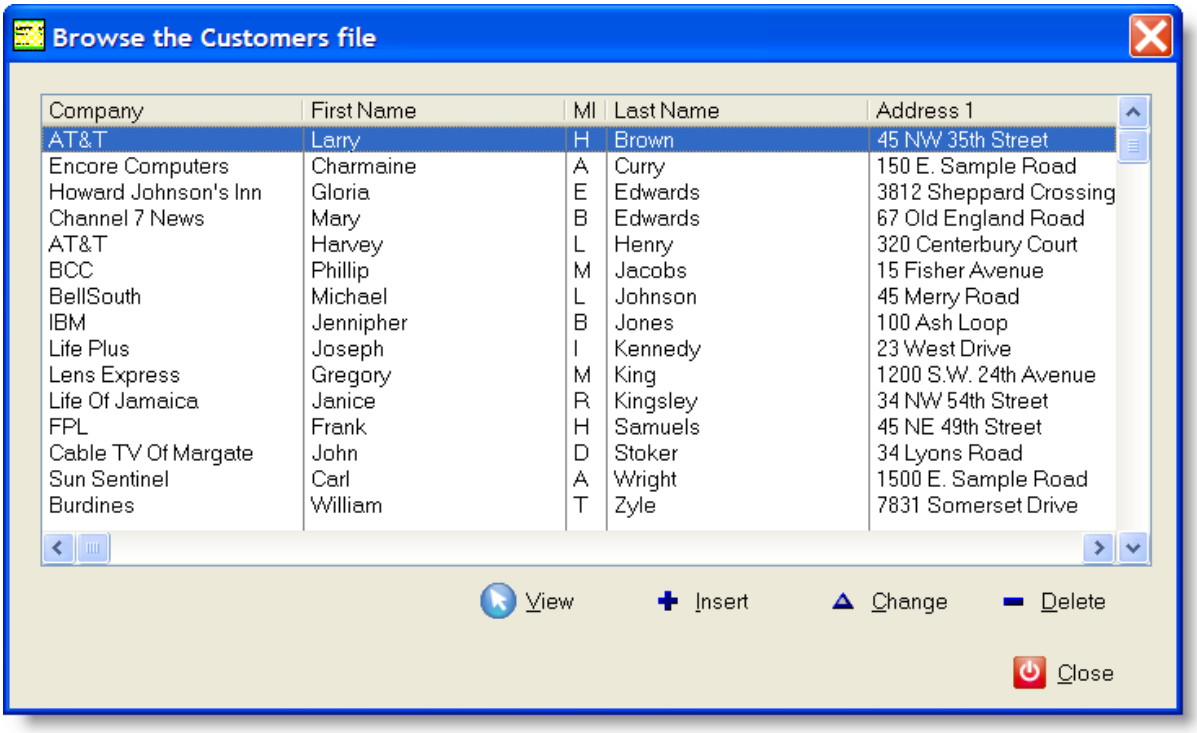


Select the "ITShowFileRecordWizard - Icetips ShowFileRecord Wizard" from the "Class IcetipsUtilityTemplate - Icetips Utility Template" and click the "Select" button.

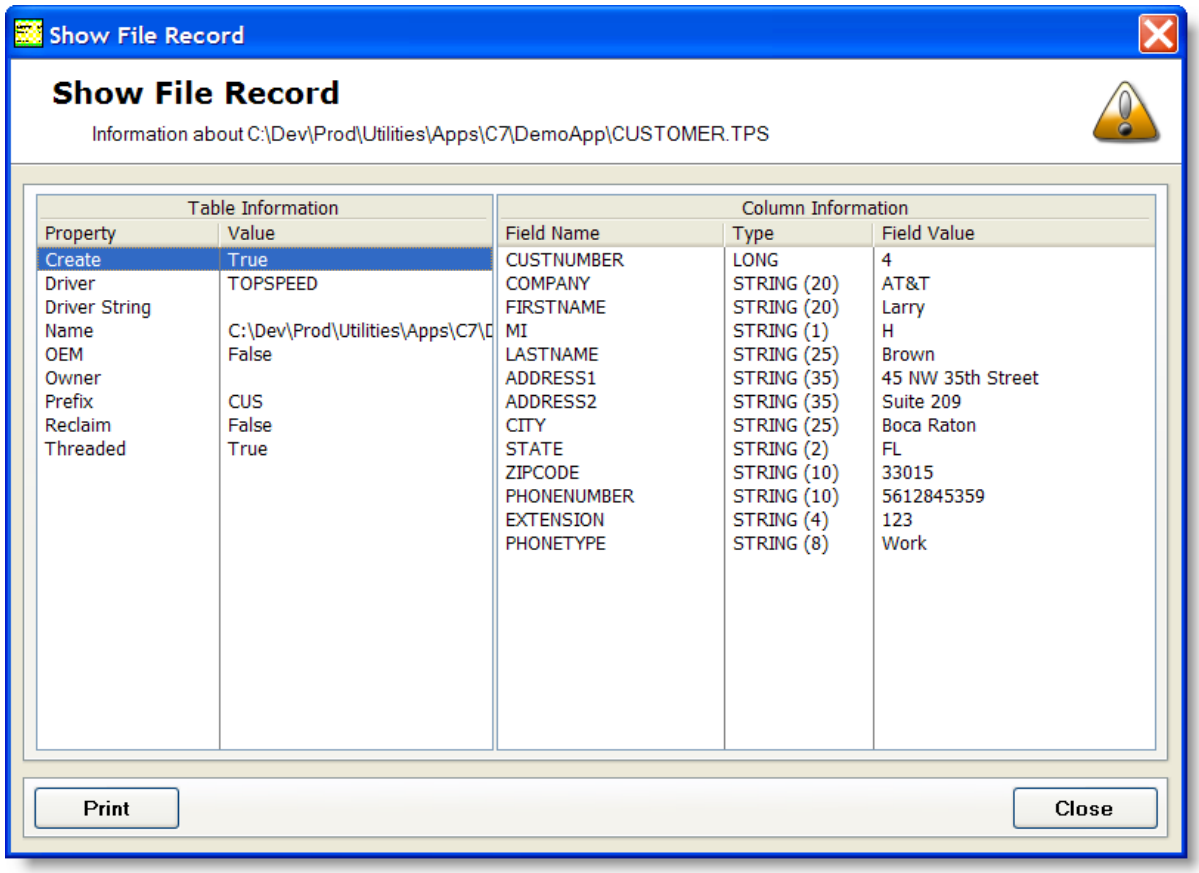


Enter the name for the procedure. ShowFileRecord is suggested as default. Once you have created the procedure you can add the "Global Call ShowRecord from Browse 439" global extension template and hook this up.

An example of the ShowFileRecord would be this browse on the customer file from the Invoice example program:



When the hotkey for the Global Call ShowRecord from Browse 439 is pressed the following window comes up.



The listbox on the left shows properties from the table structure, what driver it is, prefix, etc. You can customize the data that is shown here by modifying the LoadFileRecordInfo routine inside the procedure. It loads the table information.

The column information on the right shows each column in the table structure, what data type it is and what value is in each column.

The information can be printed, but currently it will only print the Column information, not the Table information.

The ShowFileRecord procedure is declared like this:

```
ShowFileRecordNew PROCEDURE (FILE pF, String pLabel)
```

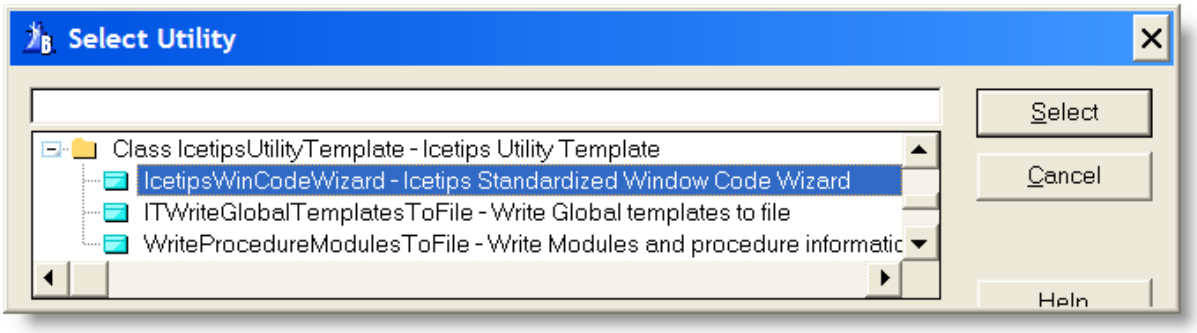
If you intend to use it in a multi-dll application, then I would suggest creating the procedure in the root/ base DLL and export it from there. Then create it as external in the apps where you need it and add the Global Call ShowRecord from Browse 439 global extension template to each app. The only thing to remember is to use the same hotkey for all the apps or this could become very confusing! You may also need to put some kind of user access restrictions on this procedure. Note, however, that the data here is all read only! There is no attempt to write it back in any way, shape or form, so no changes in data can be made with this procedure, it is for viewing only.

4.4.5 Icetips Standardized Window Code Wizard Utility Templates

This wizard template creates a procedure called WindowInitCode 514 into your application. If this procedure already exists it will attempt WindowInitCodeA, WindowInitCodeB etc. until it reaches

WindowInitCodeZ and then it will fail.

This wizard also adds an instance of the "Call procedure from all procedures 435" global extension template



The WindowInitCode 514 procedure is a source procedure that changes how your windows look. ALL the code is generated into the source procedure so it is entirely up to you how you set it up.

Use this wizard along with the "Call procedure from all procedures 435" global extension template to call this procedure from all window procedures after the window opens. This will allow you to customize all standard window controls, such as icons on buttons, fonts to use, standard settings for listboxes, etc. etc. The UtilDemo.app 514 contains a generated WindowInitCode 514 procedure that gives you an idea how it is used.

The default procedure contains setup code for various control types and base controls. Below is the full procedure code as it is created in build 1.2.2406. Note that by default the wizard sets button icons to use some of the Roma icons from www.icons-icons.com The icons referenced in the code are all included in the Utilities install in the 3rdParty\Images folder for the old Clarion IDE or accessory\images folder for the new Clarion IDE (Clarion 7, Clarion 8, etc.) Note that the code uses the GetBaseControlName 383 method which strips the instance number off the control name. So you don't need to worry about buttons that are called ?Insert:2 or ?Delete:5. Using '?Insert' and '?Delete' will find any buttons that start with '?Insert' or '?Delete' This makes it easy to change the look and feel of your application.

This wizard also adds the "Call Procedure from all procedures 435" global extension template. This adds a call to this procedure from all your window procedures right after the window has been opened giving this procedure first stab at modifying controls. On the global extension you can easily exclude procedures from calling the WindowInitCode 514 procedure. By default it excludes the first procedure as defined in the application properties.

```
!!! <summary>
!!! Generated from procedure template - Source
!!! Window Standardization Code
!!! </summary>
WindowInitCode       PROCEDURE  (Byte pIsFrame=False)!!,PROC ! Declare Procedure
! Start of "Data Section"
! [Priority 1300]

Loc:ClarionBuild     LONG                                  !
! [Priority 4000]
Loc:ListFont     String(64)
Loc:ListFontSize Byte
Loc:ListIsFlat   Byte

ITU      ITUtilityClass
I        Long
F        Long
Themed   Byte
L        Byte
```

```
CtrlName CString(101)

Loc:UseIconsOnButton Byte

FEQs    Queue
FEQ       Long
        End
X       Long

W      Long
H      Long
Loc:FEQ Long
ClearType  Byte

! [Priority 8500]

! End of "Data Section"
! Start of "Local Data After Object Declarations"
! [Priority 5000]

! End of "Local Data After Object Declarations"

  CODE
! Start of "Processed Code"
! [Priority 1800]

   Loc:ClarionBuild = 8000
! [Priority 4000]
 Themed = SYSTEM {PROP:ThemeActive}
 ClearType = ITU.UsesClearType()
 I = 0
 Loc:UseIconsOnButton = True
 If Not pIsFrame
   ! Example of setting wallpaper and window icon so the windows can be minimized.
   !0{PROP:Wallpaper} = ''
   !If 0{PROP:RESIZE} = True
   !  0{PROP:Icon}  = '~WindowIcon.ico'
   !End
 End
                              ! Example of using INI file to store these settings.
 Loc:ListFont     = 'Tahoma'  !GetIni('Window Appearance','Listbox font','Tahoma',INIFileName)
 Loc:ListFontSize = 8         !GetIni('Window Appearance','Listbox font size',9,INIFileName)
 Loc:ListIsFlat   = True      !GetIni('Window Appearance','Listbox is flat',True,INIFileName)

 ! Read FEQs into queue so they are not affected by processing.
 Loop
   I = 0{PROP:NextField,I}
   If Not I
     Break
   End
   If Not pIsFrame
     If I < 0
       Cycle
     End
   End
   FEQs.FEQ = I
   Add(FEQs)
 End

 ! Now loop through the queue and check each control.
 Loop X = 1 To Records(FEQs)
   Get(FEQs,X)
   I = FEQs.FEQ
   Loc:FEQ = I
   Case I{PROP:Type}
   Of CREATE:List
     I{PROP:FontName} = 'Tahoma'
     I{PROP:FontSize} = 9
     F = 0
     Loop
       F += 1
```

```
    If Not I{PROPLIST:Exists,F}
      Break
    End
    I{PROPLIST:HeaderCenter,F} = True
    ! Right adjust decimal adjusted fields and set offset to 2
    If I{PROPLIST:Decimal,F} = True
      I{PROPLIST:Right,F} = True
      I{PROPLIST:RightOffset,F} = 2
    End
  End
OrOf CREATE:DropList
OrOf CREATE:DropCombo
OrOf CREATE:Combo
  I{PROP:Flat}    = True
  I{PROP:Vscroll} = True
  If I{PROP:Drop} > 0
    I{PROP:VCR}        = False
  Else
    I{PROP:VCR}        = True
  End
  I {PROP:FontName} = Loc:ListFont
  I {PROP:FontSize} = Loc:ListFontSize
Of CREATE:String
OrOf CREATE:SString
  If I{PROP:Background} = COLOR:None
    I{PROP:Trn} = True
  End
Of CREATE:Prompt
  I{PROP:TRN} = True
Of CREATE:Option
OrOf CREATE:Check
  Do CheckCheckRadio
OrOf CREATE:Group
OrOf CREATE:Radio
  Do CheckCheckRadio
  I{PROP:TRN} = True
Of CREATE:Button
  ITU.GetWindowVersion()
  If ITU.UsingLargeFonts()
    ! Using Large Fonts
    If ITU.VersionPlatformID = IT_VER_PLATFORM_WIN32_NT
      I{PROP:FontName} = 'Microsoft Sans Serif'
    Else
      I{PROP:FontName} = 'Arial'
    End
    I{PROP:FontSize}  = 8
    I{PROP:FontStyle} = FONT:Regular
  Else
    ! Using Small Fonts
    If ITU.VersionPlatformID = IT_VER_PLATFORM_WIN32_NT
      I{PROP:FontName} = 'Microsoft Sans Serif'
    Else
      I{PROP:FontName} = 'Arial'
    End
    I{PROP:FontSize}  = 8
    I{PROP:FontStyle} = FONT:Regular
  End
  If Not pIsFrame
    Do SetButtonIcon
  End
Of CREATE:Text
  If Themed = False
    I {PROP:Flat} = TRUE
  End
Of CREATE:Entry
OrOf CREATE:Spin
  If Themed = False
    I {PROP:Flat} = TRUE
  End
 ! Prevent cleartype problems in entry and spin controls in versions prior to Clarion 7
```

```
    If ClearType And Loc:ClarionBuild < 7000
       I{PROP:FontName} = 'MS Sans Serif'
     End
     I{PROP:FontSize} = 8
     If I{PROP:Decimal} = True Or Instring('D',Upper(I{PROP:Text}),1,1) = 1
       I{PROP:Right}      = True
       I{PROP:RightOffset} = 1
     End
   Of CREATE:sheet
     I{PROP:Trn} = False
   Of CREATE:Panel
     I{PROP:BevelInner} = 0
     I{PROP:BevelOuter} = 1
   End
 End
 Free(FEQs)

! End of "Processed Code"
! Start of "Procedure Routines"

! [Priority 4000]
SetButtonIcon           ROUTINE
!! Remember to add any icons that are used here to the Icons.prj
!! file and recompile the project.
 Data
I  Long
 Code

 I = Loc:FEQ
 Case Upper(Clip(ITU.GetBaseControlName(I)))
 Of '?SELECT'
   I{Prop:Icon} = '~RO-Mx2_folder-down.ico'
 Of '?VIEW'
   I{Prop:Icon} = '~RO-Mx2_folder-up.ico'
 Of '?INSERT'
 OrOf '?ADD'
   I{Prop:Icon} = '~RO-Mx2_add-green.ico'
 Of '?CHANGE'
 OrOf '?EDIT'
   I{Prop:Icon} = '~RO-Mx2_notebook-pen.ico'
 Of '?DELETE'
   I{Prop:Icon} = '~RO-Mx2_delete.ico'
 Of '?CLOSE'
 OrOf '?CANCEL'
   I{Prop:Icon} = '~RO-Mx2_door-2.ico'
 Of '?OK'
   I{Prop:Icon} = '~RO-Mx2_checkmark-green.ico'
 Of '?CALENDAR'
   I{Prop:Icon} = '~RO-Mx2_calendar-month-2.ico'
 End

 If I{Prop:Icon}
   I{Prop:Left} = True
 Else
   I{Prop:Left} = False
 End
 I{Prop:Flat} = False


! [Priority 4000]
CheckCheckRadio             ROUTINE
 Data
T  String(20)
 Code

 Case Loc:FEQ
 Of CREATE:Check
 OrOf CREATE:Radio
   T = Loc:FEQ{Prop:Text}
```

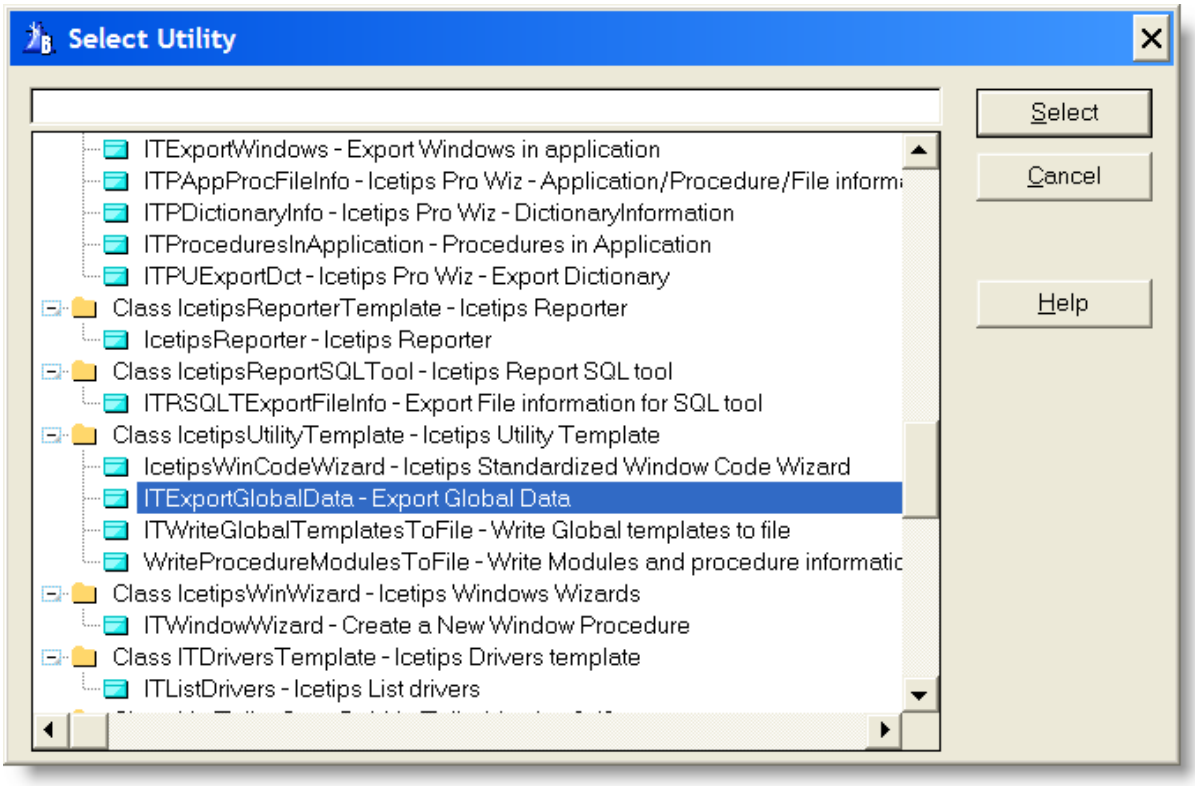
```
    If T
      If T[1] <> ' '
        Loc:FEQ{Prop:Text} = ' ' & Loc:FEQ{Prop:Text}
      End
    End
  End

! [Priority 8500]

! End of "Procedure Routines"
! Start of "Local Procedures"
! [Priority 5000]

! End of "Local Procedures"
```

4.4.6 Prepare Multi-DLL app Utility Templates

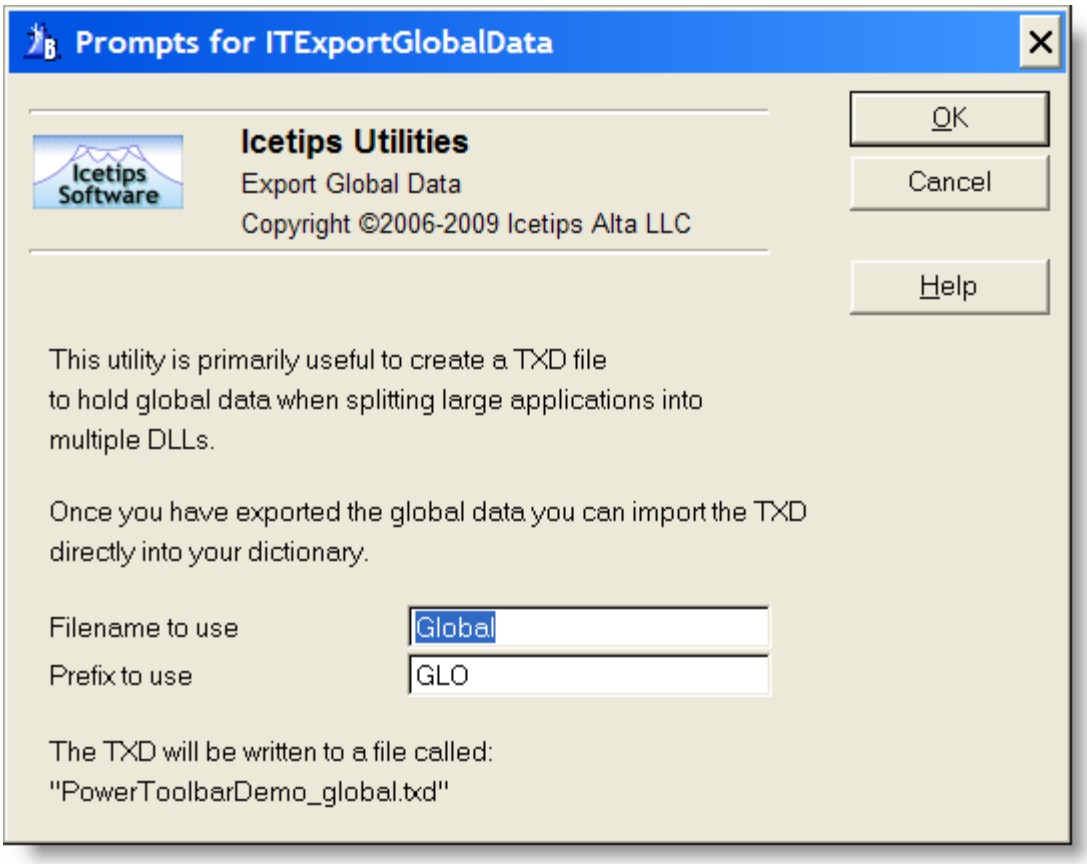
Note: This template no longer exits. Please use the "Prepare Multi-DLL app 496" template to export global data.

This template exports all global data that is in the application to a TXD file. You can then simply import the TXD file into your dictionary to get all your global data from the application. This is very handy when splitting a single application file into multiple DLLs. Then it is very convenient to keep the Global data in the dictionary and then the templates will take care of exporting them correctly.

To start, select "Application | Template Utility" from the Clarion 6 main menu or "Application | Utility Template" from the Clarion 7 menu. Alternatively you can hit Ctrl-U on the keyboard (both versions)

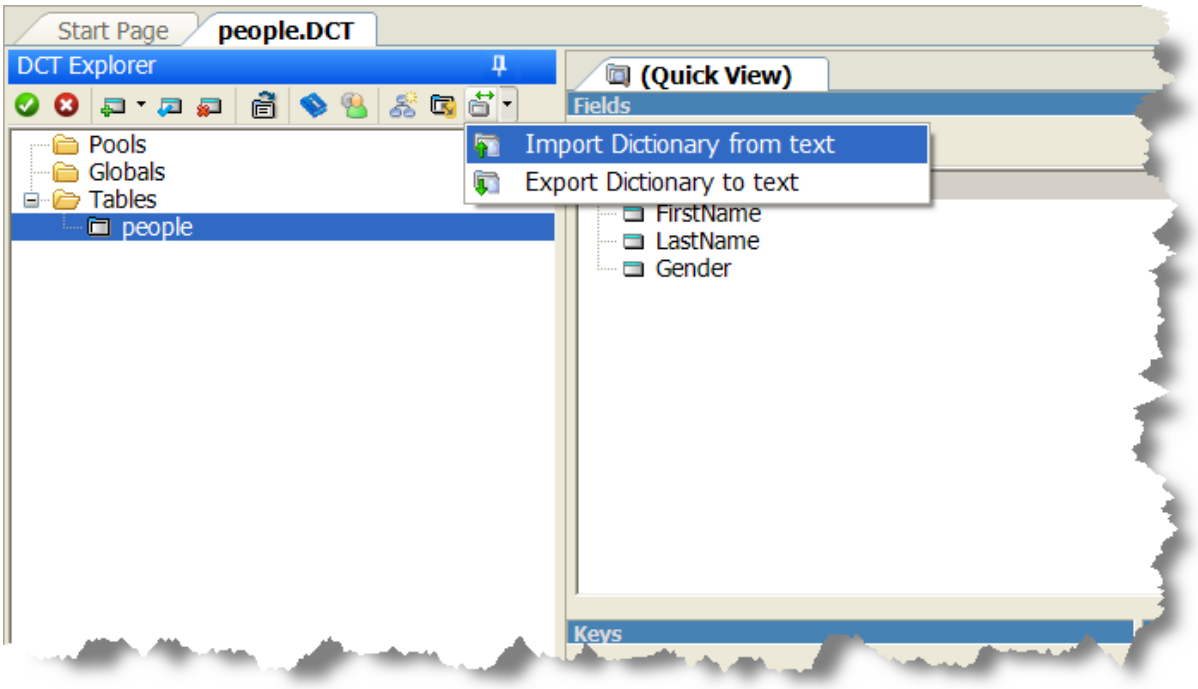


Select the "ITExportGlobalData - Export Global Data" utility template from the "IcetipsUtilityTemplate" class.



By default the filename to use is set to "Global" and the prefix to "Glo" If this does not match with what you need you can change it before you export the data. Click the OK button to export the data. It will be exported to a file that uses the same filename as the application (.app file) but with a "_global" appended to the name. This is a fully qualified TXD file so it can be imported directly, however only the label and data type are exported.

The next step is import the .txd file into your dictionary. Load the target dictionary file and select "File | Import Text" in Clarion 6. In Clarion 7 please see the screenshot below:



Example:

```
[DICTIONARY]
VERSION '1.0'
CREATED '27 SEP 2009' ' 5:49PM'
MODIFIED '27 SEP 2009' ' 5:49PM'

[FILES]
Global FILE,DRIVER('TOPSPEED'),PRE(GLO),CREATE,THREAD
!!> USAGE(Global)
Record                    RECORD
BrowseInfoShown             BYTE
Test                        STRING(20)
                          END
                        END
```

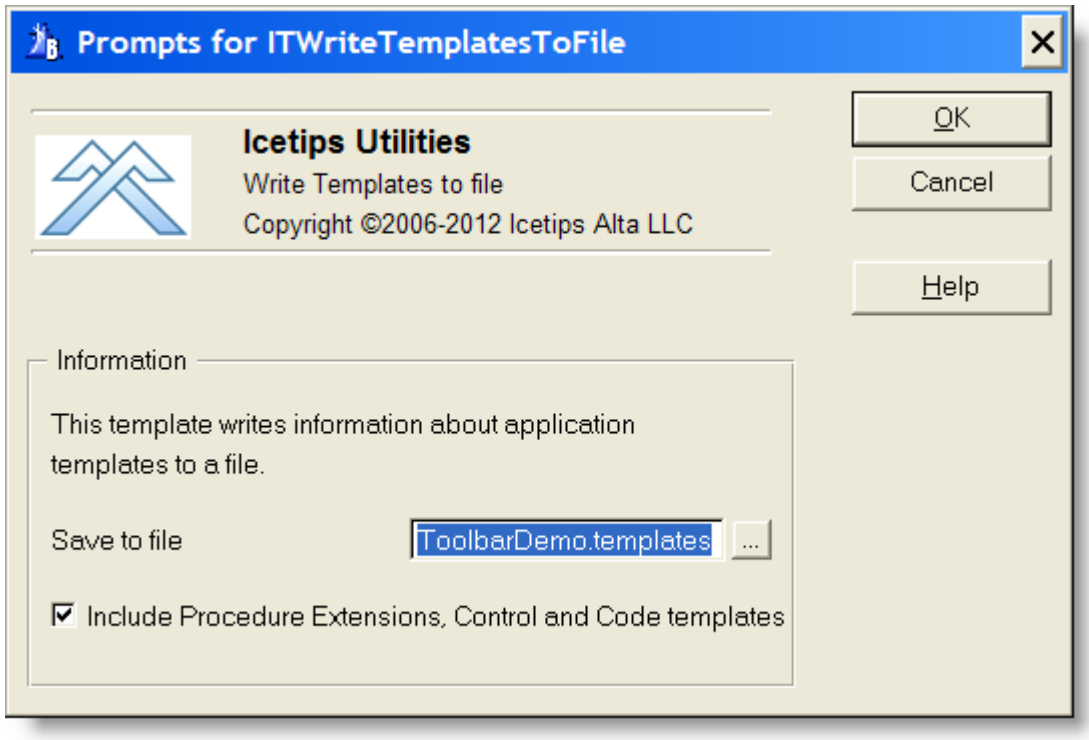
This shows a generated TXD with two Global fields in it, ready to import into a dictionary.

4.4.7 Write Used/Unused Files to file Utility Templates

Enter topic text here.

4.4.8 Write Templates to file Utility Templates

This is a powerful utility template that can write information about global extension templates and every procedure extension, control and code template used in an application to a file. Note that this does not seem to include any global code templates which do not seem to be included in the % ApplicationTemplates symbol.



Save to file Specify the file you want to write to. The template defaults to <appname>.templates

Include Procedure... If this is checked, then the template will list all procedure extension, control and code templates.

Here is an example file from the ToolbarDemo from our PowerToolbar. At the top are the Global extension templates and then it lists each procedure in alphabetical order and inside each procedure it lists the extension, control and code templates in that order. Each entry shows the template label and template chain in parenthesis. Then comes the instance number and finally the template description. This gives you a quick and easy way to see exactly what templates are used, what template chains they belong to and what instance numbers they use.

```
-------------------------------------------------------------------------------
Application: PowerToolbarDemo.app  List created on June 23, 2012 at 12:59:31
-------------------------------------------------------------------------------

GLOBAL EXTENSION TEMPLATES:                                  Inst     Description
  PowerToolbarGlobal(PowerToolbar)                              1     Icetips PowerToolbar Globa
  XPTheme(XPTheme)                                              2     Icetips PowerXP-Theme 3
  ITUtilitiesGlobal(IcetipsUtilityTemplate)                     3     Icetips Utilities Classes
  ITGlobalLimitProgramInstances(IcetipsUtilityTemplate)         4     Limit Program Instance

-------------------------------------------------------------------------------
BrowseDemo:                                                  Inst     Description

  EXTENSION TEMPLATES:
    PowerToolbarMDIClient(PowerToolbar)                         3     Icetips PowerToolbar MDI C

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                           5     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    BrowseBox(ABC)                                              1     Browse on SampleTable
      ?List (LIST)
```

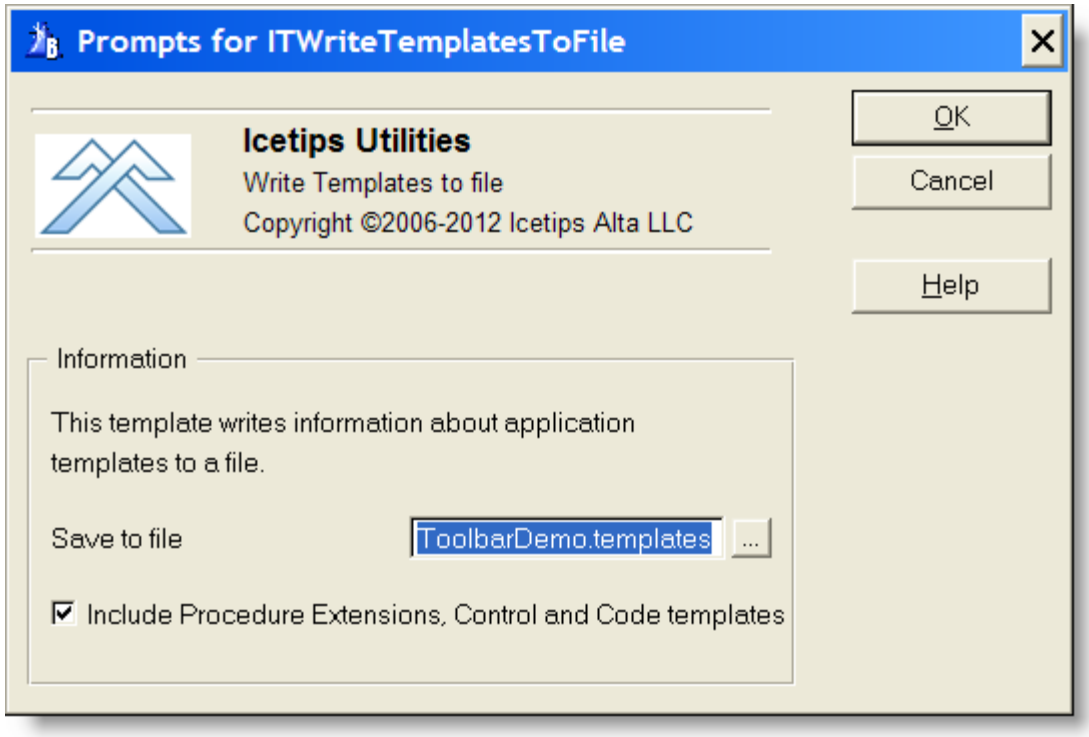
```
    BrowseViewButton(ABC)                                  4     View a Record from Browse
      ?View (BUTTON)
    BrowseUpdateButtons(ABC)                               2     Update a Record from Brows
      ?Insert (BUTTON)
      ?Change (BUTTON)
      ?Delete (BUTTON)

  CODE TEMPLATES:
    EmbedInformation(ABCFree)                              6     Embed: Display EMBED() inf

-------------------------------------------------------------------------------
BrowseDemoForm:                                          Inst    Description

  CONTROL TEMPLATES:
    SaveButton(ABC)                                        1     Update SampleTable record
      ?OK (BUTTON)
    CancelButton(ABC)                                      2     Cancel the Current Operati
      ?Cancel (BUTTON)

-------------------------------------------------------------------------------
ColorDemo:                                               Inst    Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                      1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    PowerToolbarControl(PowerToolbar)                      2     Icetips PowerToolbar Contr
      ?Toolbar1:2 (IMAGE)
    PowerToolbarControl(PowerToolbar)                      3     Icetips PowerToolbar Contr
      ?Toolbar1:3 (IMAGE)
    PowerToolbarControl(PowerToolbar)                      4     Icetips PowerToolbar Contr
      ?Toolbar1:4 (IMAGE)
    PowerToolbarControl(PowerToolbar)                      5     Icetips PowerToolbar Contr
      ?Toolbar1:5 (IMAGE)

-------------------------------------------------------------------------------
ControlTypesDemo:                                        Inst    Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                      1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
IExplorerDemo:                                           Inst    Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                      1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
IconsDemo:                                               Inst    Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                      1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
Main:                                                    Inst    Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                      1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
MenuTest:                                                Inst    Description

-------------------------------------------------------------------------------
```

```
NotepadDemo:                                                Inst     Description

  EXTENSION TEMPLATES:
    XPThemeWindow(XPTheme)                                     4     Icetips PowerXP-Theme Wind
    PowerToolbarMDIClient(PowerToolbar)                        5     Icetips PowerToolbar MDI C
    WindowResize(ABC)                                          1     Allows controls to be resi

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                          6     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    RTFTextControl(ABC)                                        2     RTF Text Control(2)
      ?RTFTextBox (TEXT)
    RTFToolbar(ABC)                                            3     RTF Toolbar
      ?RTFToolNew (BUTTON)
      ?RTFToolOpen (BUTTON)
      ?RTFToolSave (BUTTON)
      ?RTFToolPrint (BUTTON)
      ?RTFToolFind (BUTTON)
      ?RTFToolFindAndReplace (BUTTON)
      ?RTFToolBold (CHECK)
      ?RTFToolCut (BUTTON)
      ?RTFToolCopy (BUTTON)
      ?RTFToolPaste (BUTTON)
      ?RTFToolUndo (BUTTON)
      ?RTFToolRedo (BUTTON)
      ?RTFToolTabs (BUTTON)
      ?RTFToolPara (BUTTON)
      ?RTFToolAlignment (OPTION)
      ?RTFToolAlignmentLeft (RADIO)
      ?RTFToolAlignmentCenter (RADIO)
      ?RTFToolAlignmentRight (RADIO)
      ?RTFToolAlignmentJust (RADIO)
      ?RTFToolBullets (CHECK)
      ?RTFToolBulletStyle (LIST)
      ?RTFToolItalic (CHECK)
      ?RTFToolUnderline (CHECK)
      ?RTFToolFontColor (LIST)
      ?RTFToolFontBkColor (LIST)
      ?RTFToolFontName (LIST)
      ?RTFToolFontSize (LIST)
      ?RTFToolFontScript (LIST)

-------------------------------------------------------------------------------
OutlookDemo:                                                Inst     Description

  EXTENSION TEMPLATES:
    XPThemeWindow(XPTheme)                                     2     Icetips PowerXP-Theme Wind

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                          1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
StylesDemo:                                                 Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                          1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    PowerToolbarControl(PowerToolbar)                          2     Icetips PowerToolbar Contr
      ?Toolbar1:2 (IMAGE)
    PowerToolbarControl(PowerToolbar)                          3     Icetips PowerToolbar Contr
      ?Toolbar1:3 (IMAGE)
    PowerToolbarControl(PowerToolbar)                          4     Icetips PowerToolbar Contr
      ?Toolbar1:4 (IMAGE)

-------------------------------------------------------------------------------
```

```
TabbedViewDemo:                                                  Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                              1      Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
Welcome:                                                         Inst     Description

  CODE TEMPLATES:
    EmbedInformation(ABCFree)                                      1      Embed: Display EMBED() inf
```

4.4.9 Write Templates to file compact Utility Templates

This is a powerful utility template that can write information about global extension templates and every procedure extension, control and code template used in an application to a file. Note that this does not seem to include any global code templates which do not seem to be included in the % ApplicationTemplates symbol.



Save to file Specify the file you want to write to. The template defaults to <appname>.templates

Include Procedure... If this is checked, then the template will list all procedure extension, control and code templates.

Here is an example file from the ToolbarDemo from our PowerToolbar. At the top are the Global extension templates and then it lists each procedure in alphabetical order and inside each procedure it lists the extension, control and code templates in that order. Each entry shows the template label and template chain in parenthesis. Then comes the instance number and finally the template description. This gives you a quick and easy way to see exactly what templates are used, what template chains

they belong to and what instance numbers they use.

```
-------------------------------------------------------------------------------
Application: PowerToolbarDemo.app  List created on June 23, 2012 at 12:59:31
-------------------------------------------------------------------------------

GLOBAL EXTENSION TEMPLATES:                                  Inst     Description
  PowerToolbarGlobal(PowerToolbar)                              1     Icetips PowerToolbar Globa
  XPTheme(XPTheme)                                              2     Icetips PowerXP-Theme 3
  ITUtilitiesGlobal(IcetipsUtilityTemplate)                     3     Icetips Utilities Classes
  ITGlobalLimitProgramInstances(IcetipsUtilityTemplate)         4     Limit Program Instance

-------------------------------------------------------------------------------
BrowseDemo:                                                  Inst     Description

  EXTENSION TEMPLATES:
    PowerToolbarMDIClient(PowerToolbar)                         3     Icetips PowerToolbar MDI C

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                           5     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    BrowseBox(ABC)                                              1     Browse on SampleTable
      ?List (LIST)
    BrowseViewButton(ABC)                                       4     View a Record from Browse
      ?View (BUTTON)
    BrowseUpdateButtons(ABC)                                    2     Update a Record from Brows
      ?Insert (BUTTON)
      ?Change (BUTTON)
      ?Delete (BUTTON)

  CODE TEMPLATES:
    EmbedInformation(ABCFree)                                   6     Embed: Display EMBED() inf

-------------------------------------------------------------------------------
BrowseDemoForm:                                              Inst     Description

  CONTROL TEMPLATES:
    SaveButton(ABC)                                             1     Update SampleTable record
      ?OK (BUTTON)
    CancelButton(ABC)                                           2     Cancel the Current Operati
      ?Cancel (BUTTON)

-------------------------------------------------------------------------------
ColorDemo:                                                   Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                           1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    PowerToolbarControl(PowerToolbar)                           2     Icetips PowerToolbar Contr
      ?Toolbar1:2 (IMAGE)
    PowerToolbarControl(PowerToolbar)                           3     Icetips PowerToolbar Contr
      ?Toolbar1:3 (IMAGE)
    PowerToolbarControl(PowerToolbar)                           4     Icetips PowerToolbar Contr
      ?Toolbar1:4 (IMAGE)
    PowerToolbarControl(PowerToolbar)                           5     Icetips PowerToolbar Contr
      ?Toolbar1:5 (IMAGE)

-------------------------------------------------------------------------------
ControlTypesDemo:                                            Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                           1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
IExplorerDemo:                                               Inst     Description
```

```
  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                       1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
IconsDemo:                                                Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                       1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
Main:                                                     Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                       1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
MenuTest:                                                 Inst     Description

-------------------------------------------------------------------------------
NotepadDemo:                                              Inst     Description

  EXTENSION TEMPLATES:
    XPThemeWindow(XPTheme)                                  4     Icetips PowerXP-Theme Wind
    PowerToolbarMDIClient(PowerToolbar)                     5     Icetips PowerToolbar MDI C
    WindowResize(ABC)                                       1     Allows controls to be resi

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                       6     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    RTFTextControl(ABC)                                     2     RTF Text Control(2)
      ?RTFTextBox (TEXT)
    RTFToolbar(ABC)                                         3     RTF Toolbar
      ?RTFToolNew (BUTTON)
      ?RTFToolOpen (BUTTON)
      ?RTFToolSave (BUTTON)
      ?RTFToolPrint (BUTTON)
      ?RTFToolFind (BUTTON)
      ?RTFToolFindAndReplace (BUTTON)
      ?RTFToolBold (CHECK)
      ?RTFToolCut (BUTTON)
      ?RTFToolCopy (BUTTON)
      ?RTFToolPaste (BUTTON)
      ?RTFToolUndo (BUTTON)
      ?RTFToolRedo (BUTTON)
      ?RTFToolTabs (BUTTON)
      ?RTFToolPara (BUTTON)
      ?RTFToolAlignment (OPTION)
      ?RTFToolAlignmentLeft (RADIO)
      ?RTFToolAlignmentCenter (RADIO)
      ?RTFToolAlignmentRight (RADIO)
      ?RTFToolAlignmentJust (RADIO)
      ?RTFToolBullets (CHECK)
      ?RTFToolBulletStyle (LIST)
      ?RTFToolItalic (CHECK)
      ?RTFToolUnderline (CHECK)
      ?RTFToolFontColor (LIST)
      ?RTFToolFontBkColor (LIST)
      ?RTFToolFontName (LIST)
      ?RTFToolFontSize (LIST)
      ?RTFToolFontScript (LIST)

-------------------------------------------------------------------------------
```

```
OutlookDemo:                                                    Inst     Description

  EXTENSION TEMPLATES:
    XPThemeWindow(XPTheme)                                        2     Icetips PowerXP-Theme Wind

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                             1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
StylesDemo:                                                     Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                             1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)
    PowerToolbarControl(PowerToolbar)                             2     Icetips PowerToolbar Contr
      ?Toolbar1:2 (IMAGE)
    PowerToolbarControl(PowerToolbar)                             3     Icetips PowerToolbar Contr
      ?Toolbar1:3 (IMAGE)
    PowerToolbarControl(PowerToolbar)                             4     Icetips PowerToolbar Contr
      ?Toolbar1:4 (IMAGE)

-------------------------------------------------------------------------------
TabbedViewDemo:                                                 Inst     Description

  CONTROL TEMPLATES:
    PowerToolbarControl(PowerToolbar)                             1     Icetips PowerToolbar Contr
      ?Toolbar1 (IMAGE)

-------------------------------------------------------------------------------
Welcome:                                                        Inst     Description

  CODE TEMPLATES:
    EmbedInformation(ABCFree)                                     1     Embed: Display EMBED() inf
```

4.4.10 Write Icons and Images to File — Utility Templates

This template allows you to export a list of all icons and images from an application. It both lists the images for each procedure and then at the end a list of all the image files in alphabetical order without duplicates. Below is an example list of all the icons and images in the Build Automator main executable application.

```
-------------------------------------------------------------------------------
Application: BuildAutomator.app  List created on August 21, 2014 at 13:10:37
-------------------------------------------------------------------------------

Icons/Images:

ActionEditor:
  Window:

    ?ExploreVATNButton            ICON(ICON:None)
    ?ITCHeaderImage               IMAGE

ActionItemProperties:
  Window:

    ?ITCHeaderImage               IMAGE('GL-Bu2_info.ico')
```

```
AutomatorOptions:
  Window:

    ?ITCHeaderImage                IMAGE('RO-Mx1_options-list.ico')
    ?LookupFile                    ICON('RO-Mx2_folder.ico')

CreateProjectShortcut:
  Window:

    ?ITCHeaderImage                IMAGE('Automator Icon.ico')

ITPreviewer:
  ITPreviewWindow:

    ITPreviewWindow                ICON(Icon:Print)
    ?PrintAllButton                ICON('ITprint.ico')
    ?PrintOneButton                ICON('ITprint1b.ico')
    ?CancelButton                  ICON('ITnoprintb.ico')
    ?JumpFirstButton               ICON('ITfirstb.ico')
    ?JumpPrevButton                ICON('ITprevpageb.ico')
    ?JumpNextButton                ICON('ITnextpageb.ico')
    ?JumpLastButton                ICON('ITlastb.ico')
    ?ITP:ShowPageList              ICON('ITpagedropb.ico')
    ?ZoomInButton                  ICON('ITzoominb.ico')
    ?ZoomOutButton                 ICON('ITzoomoutb.ico')
    ?ZoomToPageWidth               ICON('ITpagewb.ico')
    ?ZoomToPageHeight              ICON('ITpagehb.ico')
    ?Zoom100Percent                ICON('ITpagefullb.ico')
    ?SearchButton                  ICON('ITsearchb.ico')
    ?TagAllFound                   ICON('ITmark.ico')
    ?UntagAllFound                 ICON('ITunmark.ico')
    ?TogglePrintButton             ICON('ITmark.ico')
    ?PageImage                     IMAGE

InstallRegistrationKey:
  Window:
    'windowbackground.bmp'
    Window                         WALLPAPER('windowbackground.bmp')
    Window                         ICON('Automator Icon.ico')
    ?Button3                       ICON('RO-Mx2_circle-green-cur-dollar.ico')
    ?ITCHeaderImage                IMAGE('RO-Mx2_key.ico')

LoadStartup:
  Window:
    'toolbarbackground55.bmp'
    Window                         WALLPAPER('toolbarbackground55.bmp')

Main:
  AppFrame:
    'Automator wallpaper.bmp'
    AppFrame                       WALLPAPER('Automator wallpaper.bmp')
    AppFrame                       ICON('Automator Icon.ico')
    ?FileNew                       ICON('RO-Mx1_folder-new.ico')
    ?FileOpen                      ICON('RO-Mx1_folder-open.ico')
    ?FileSelectFilefromList        ICON('RO-Mx1_folder-view.ico')
    ?FileSave                      ICON('RO-Mx1_diskette-blue-2.ico')
    ?FilePrint                     ICON('RO-Mx1_print-1.ico')
    ?ToolsActionEditor             ICON('RO-Mx1_tool-wrench.ico')
    ?ToolsOptions                  ICON('RO-Mx1_options-list.ico')
    ?HelpVisitourwebsite           ICON('Automator Icon.ico')
    ?HelpCheckforupdates           ICON('RO-Mx1_internet-america.ico')
    ?HelpOnlineSupport             ICON('RO-Mx1_circle-help-2.ico')
    ?HelpReportProblems            ICON('RO-Mx1_bandage.ico')
    ?Item26                        ICON('RO-Mx2_key.ico')
    ?NewFileButton                 ICON('RO-Mx1_folder-new.ico')
```

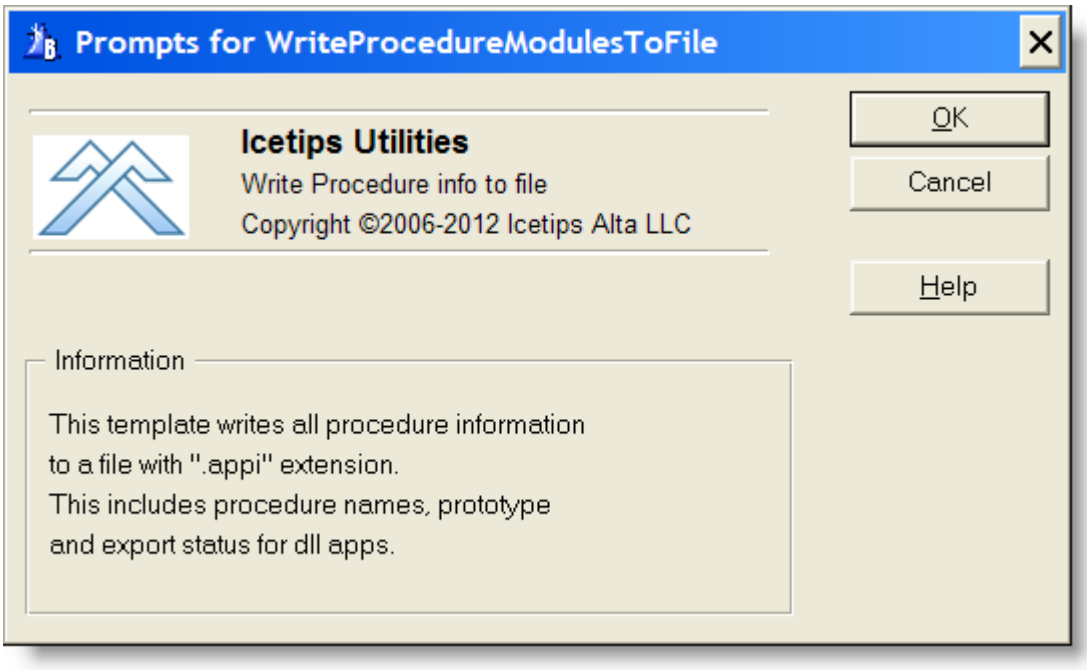
```
    ?OpenButton                 ICON('RO-Mx1_folder-open.ico')
    ?SaveFileButton             ICON('RO-Mx1_diskette-blue-2.ico')
    ?PrintButton                ICON('RO-Mx1_print-1.ico')
    ?ActionEditor               ICON('RO-Mx1_tool-wrench.ico')
    ?OptionsButton              ICON('RO-Mx1_options-list.ico')
    ?PicklistButton             ICON('RO-Mx1_folder-view.ico')
    ?WebHomeButton              ICON('Automator Icon.ico')
    ?WebGoToAccount             ICON('RO-Mx1_users-1.ico')
    ?WebSupportButton           ICON('RO-Mx1_circle-help-2.ico')
    ?CheckForUpdatesButton      ICON('RO-Mx1_internet-america.ico')
    ?Toolbar1                   IMAGE
    ?Close                      ICON('RO-Mx1_door-2.ico')

MaintenancePlan:
  Window:

    ?BuyButton                  ICON('RO-Mx2_circle-green-cur-dollar.ico')
    ?ITCHeaderImage             IMAGE('RO-Mx1_ribbon-yellow.ico')

OpenSupportChannel:
  Window:

    ?Image1                     IMAGE('wormhole.gif')

PickList:
  Window:
    'windowbackground.bmp'
    Window                      WALLPAPER('windowbackground.bmp')
    ?ClearFilterButton          ICON('RO-Mx1_circle-red-cancel.ico')
    ?ClearListButton            ICON('RO-Mx1_trash.ico')
    ?NewProjectButton           ICON('RO-Mx1_folder-new.ico')
    ?SelectButton               ICON('RO-Mx1_folder-down.ico')
    ?OpenProjectButton          ICON('RO-Mx1_folder-diskette.ico')
    ?Image2                     IMAGE('bluegrad.bmp')

ProjectItemProperties:
  Window:

    ?ITCHeaderImage             IMAGE('GL-Bu2_info.ico')

ProjectProperties:
  Window:

    ?ITCHeaderImage             IMAGE('GL-Bu2_info.ico')

ProjectWindow:
  Window:

    ?Toolbar1                   IMAGE
    ?ProjectPropButtone         ICON('RO-Mx1_notebook-pen.ico')
    ?ViewLogfileButton          ICON('RO-Mx1_doc-view.ico')
    ?VariablesButton            ICON('RO-Mx1_db-list-table.ico')
    ?ExecuteFromStartButton     ICON('arrowhead-check-right.ico')
    ?ExecuteButton              ICON('arrowhead-green.ico')
    ?ExecuteProjectItemButton   ICON('arrowhead-list.ico')
    ?ExecuteFromItemButton      ICON('arrowhead-check-left.ico')
    ?ExecuteItemButton          ICON('arrowhead-boxed.ico')
    ?StepThroughButton          ICON('RO-Mx2_circle-blue-vcr-pause.ico')
    ?MoveUpButton               ICON('RO-Mx1_arrow-green-up.ico')
    ?MoveDownButton             ICON('RO-Mx1_arrow-green-down.ico')
    ?PrintProjectButton         ICON('RO-Mx2_printer.ico')
    ?PrintCheckListButton       ICON('RO-Mx1_doc-check-green.ico')
    ?CloseButton                ICON('RO-Mx1_x-red.ico')
    ?ProjectImage               IMAGE('toolbarbackground40.bmp')
    ?ActionItemImage            IMAGE('toolbarbackground40.bmp')
    ?ActionImage                IMAGE('toolbarbackground40.bmp')
```

```
SelectGenerateLanguage:
  Window:


SplashWindow:
  Window:
    'Automator Splash 2014.bmp'
    Window                          WALLPAPER('Automator Splash 2014.bmp')
    ?Close                          ICON('RO-Mx2_door-2.ico')

UpdateActionFile:
  Window:

    ?LookupFile:2                   ICON(ICON:Open)
    ?LookupFile                     ICON(ICON:Open)
    ?ITCHeaderImage                 IMAGE

UpdateActions:
  Window:

    ?ITCHeaderImage                 IMAGE

UpdateServers:
  Window:


UpdateVendorFile:
  Window:

    ?TestEmailButton                ICON('RO-Mx2_envelope-closed.ico')
    ?TestWebButton                  ICON('RO-Mx2_globe-america.ico')
    ?ITCHeaderImage                 IMAGE

Variables:
  Window:


ViewLogFile:
  Window:

    ?ITCHeaderImage                 IMAGE('GL-Bu2_document-text.ico')

ViewLogFileNotMDI:
  Window:

    ?ITCHeaderImage                 IMAGE('GL-Bu2_document-text.ico')


List of all Unique Image Names in the application
-------------------------------------------------

  'Automator Icon.ico'
  'GL-Bu2_document-text.ico'
  'GL-Bu2_info.ico'
  'ITfirstb.ico'
  'ITlastb.ico'
  'ITmark.ico'
  'ITnextpageb.ico'
  'ITnoprintb.ico'
  'ITpagedropb.ico'
  'ITpagefullb.ico'
  'ITpagehb.ico'
  'ITpagewb.ico'
  'ITprevpageb.ico'
  'ITprint.ico'
```

```
'ITprint1b.ico'
'ITsearchb.ico'
'ITunmark.ico'
'ITzoominb.ico'
'ITzoomoutb.ico'
'RO-Mx1_arrow-green-down.ico'
'RO-Mx1_arrow-green-up.ico'
'RO-Mx1_bandage.ico'
'RO-Mx1_circle-help-2.ico'
'RO-Mx1_circle-red-cancel.ico'
'RO-Mx1_db-list-table.ico'
'RO-Mx1_diskette-blue-2.ico'
'RO-Mx1_doc-check-green.ico'
'RO-Mx1_doc-view.ico'
'RO-Mx1_door-2.ico'
'RO-Mx1_folder-diskette.ico'
'RO-Mx1_folder-down.ico'
'RO-Mx1_folder-new.ico'
'RO-Mx1_folder-open.ico'
'RO-Mx1_folder-view.ico'
'RO-Mx1_internet-america.ico'
'RO-Mx1_notebook-pen.ico'
'RO-Mx1_options-list.ico'
'RO-Mx1_print-1.ico'
'RO-Mx1_ribbon-yellow.ico'
'RO-Mx1_tool-wrench.ico'
'RO-Mx1_trash.ico'
'RO-Mx1_users-1.ico'
'RO-Mx1_x-red.ico'
'RO-Mx2_circle-blue-vcr-pause.ico'
'RO-Mx2_circle-green-cur-dollar.ico'
'RO-Mx2_door-2.ico'
'RO-Mx2_envelope-closed.ico'
'RO-Mx2_folder.ico'
'RO-Mx2_globe-america.ico'
'RO-Mx2_key.ico'
'RO-Mx2_printer.ico'
'arrowhead-boxed.ico'
'arrowhead-check-left.ico'
'arrowhead-check-right.ico'
'arrowhead-green.ico'
'arrowhead-list.ico'
'bluegrad.bmp'
'toolbarbackground40.bmp'
'wormhole.gif'
ICON:None
ICON:Open
```

4.4.11 Write Modules and procedure information to File — Utility Templates

This utility template writes information about each procedure to a <appname>.appi file. The resulting file is a comma separated file with information about each procedure, such as Line number, application type (EXE, DLL, LIB), dependency (true if global data is external, false if global data is not external), procedure name, procedure prototype, procedure template, module name and if the procedure is external or exported. This is invaluable when working with multi-dll systems when creating external procedures as it shows the prototype etc. It is also very handy when doing code searches as it shows the module name so it's easy to open the module even if the Clarion IDE is not open. There are no prompts on this template.



For our PowerToolbar multi-dll test app that we compile before each release, this template creates the following data into the PowerToolbarDemo.appi file:

```
"Line","Type","Dependent","Procedure","Prototype","Template","Module","Export","External"
"1","EXE","1","BrowseDemo","","Browse","Power004.clw","0","0"
"2","EXE","1","BrowseDemoForm","","Form","Power006.clw","0","0"
"3","EXE","1","ColorDemo","","Window","Power011.clw","0","0"
"4","EXE","1","ControlTypesDemo","","Window","Power003.clw","0","0"
"5","EXE","1","GetProgramID","(String pID),String","Source","Power015.clw","0","0"
"6","EXE","1","IExplorerDemo","","Window","Power009.clw","0","0"
"7","EXE","1","IconsDemo","","Window","Power007.clw","0","0"
"8","EXE","1","Main","","Frame","Power001.clw","0","0"
"9","EXE","1","MenuTest","","Window","Power014.clw","0","0"
"10","EXE","1","NotepadDemo","","Window","Power010.clw","0","0"
"11","EXE","1","OutlookDemo","","Window","Power008.clw","0","0"
"12","EXE","1","StylesDemo","","Window","Power002.clw","0","0"
"13","EXE","1","TabbedViewDemo","","Window","Power013.clw","0","0"
"14","EXE","1","Welcome","","Window","Power012.clw","0","0"
```

Part V

Chapter 5 - Learning Icetips Utilities

5 Learning Icetips Utilities

We hope that this documentation helps you in learning how to use the Icetips Utilities. We have included some example applications 513 and we are going to add more as we progress in documenting.

Video tutorials will be added to our Youtube website at http://www.youtube.com/IcetipsVideos and we hope they will help you. We plan to have the first one up there on May 7, 2012. If you have suggestions about tutorials, please do not hesitate to contact us.

5.1 Example Applications

Enter topic text here.

5.1.1 CoreClassDemo.app Example Applications

The CoreClassDemo.app is dedicated to the CoreClass 52 only. By default the UtilDemo.app Example app is installed in your Clarion\3rdParty\Examples\ITUtilities folder.

Procedure	Demonstrates
CoreClassDemo	Shows how to create GUIDs with CreateGUID 60, retrieve and change file attributes with GetFileAttrib 64 and SetFileAttrib 78. It also shows how to get temp filenames with GetTempFile 68 and the default folder for temp files with GetTempFolder 69. Also shows GetFilePart 65 and RemoveBackSlash 76
SplashWindow	Shows a splash window.

5.1.2 WindowsClassDemo.app Example Applications

The WindowsClassDemo.app is dedicated to the WindowsClass 365 only. By default the WindowsClassDemo.app Example app is installed in your Clarion\3rdParty\Examples\ITUtilities folder. The application has 5 procedures:

Procedure	Demonstrates
Main	Shows how to color the background of an appframe with SetWindowColor 408
TestWindowsClass	Shows how to use a number of the methods in the WindowsClass, such as enumerate child and top windows, search window titles, pinpoint popup menus, change the window to use toolbox caption and set the window color. This procedure is not done and we will be modifying and adding to it as we complete the demo.
TestEnumChildWindows	Enumerates all windows that are child windows of the parent handle that is passed to the procedure. This includes controls that are on the window being enumerated.
TestEnumTopWindows	This enumerates all top windows (parent windows) that are running.
SplashWindow	Shows a splash window.

5.1.2.1 Procedures Example Applications - WindowsClassDemo.app

Enter topic text here.

5.1.2.1.1 TestEnumTopWindows

Enter topic text here.

5.1.2.1.2 TestEnumChildWindows

Enter topic text here.

5.1.3 UtilDemo.app Example Applications

The Example application contains procedures that demonstrate the use of the various classes and templates in the Icetips Utilities. By default the UtilDemo.app Example app is installed in your Clarion\3rdParty\Examples\ITUtilities folder.

5.1.3.1 Procedures Example Applications - UtilDemo.app

Enter topic text here.

5.1.3.1.1 WindowInitCode

This procedure is an example of how you can standardize various parts of your application from a single procedure. This procedure is called from all window procedure by using the "Call procedure from all procedures 435" Global extension template. This procedure uses a local queue to store all control FEQs in it. This is done to prevent any possible problems with certain properties altering the processing order of controls, which can happen when certain properties are changed. This is done with the following example code:

```
Loop
  I = 0{PROP:NextField,I}
  If Not I
    Break
  End
  If Not pIsFrame
    If I < 0
      Cycle
    End
  End
  FEQs.FEQ = I
  Add(FEQs)
End
```

Then this queue is processed and each controls is checked for it's type:

```
Loop X = 1 To Records(FEQs)
  Get(FEQs,X)
  I = FEQs.FEQ
  Loc:FEQ = I
  Case I{PROP:Type}
  ...
End
```

Each control type get's specific settings, such as this section for various listbox types:

```
Of CREATE:List
  I{PROP:FontName} = 'Tahoma'
  I{PROP:FontSize} = 9
  F = 0
  Loop
    F += 1
    If Not I{PROPLIST:Exists,F}
      Break
    End
    I{PROPLIST:HeaderCenter,F} = True
    ! Right adjust decimal adjusted fields and set offset to 2
    If I{PROPLIST:Decimal,F} = True
      I{PROPLIST:Right,F} = True
      I{PROPLIST:RightOffset,F} = 2
    End
  End
OrOf CREATE:DropList
OrOf CREATE:DropCombo
OrOf CREATE:Combo
  I{PROP:Flat}    = True
  I{PROP:Vscroll} = True
  If I{PROP:Drop} > 0
    I{PROP:VCR}        = False
  Else
    I{PROP:VCR}        = True
  End
  I {PROP:FontName} = Loc:ListFont
  I {PROP:FontSize} = Loc:ListFontSize
```

And this part that deals with entry and spin fields:

```
Of CREATE:Entry
OrOf CREATE:Spin
  If Themed = False
    I {PROP:Flat} = TRUE
  End
  If ClearType  And Loc:ClarionBuild < 7000
    I{PROP:FontName} = 'MS Sans Serif'
  End
  I{PROP:FontSize} = 8
  If I{PROP:Decimal} = True Or Instring('D',Upper(I{PROP:Text}),1,1) = 1
    I{PROP:Right}      = True
    I{PROP:RightOffset} = 1
  End
```

In this case it changes the font name if the user is using ClearType and if this is compiled with Clarion prior to version 7.0 Clarion 6.3 has problems with TrueType fonts when used with ClearType, so we change the font to MS Sans Serif which is a bitmap font and works fine. May not look the best, but it beats chopped up character which you will get if you use TrueType fonts with ClearType!

See also:
Call procedure from all procedures 435

5.1.3.1.2 TestTemplate

This procedure is used to demonstrate some of the templates that are included in the Icetips Utilities.

Add Procedures To Queue 417
Sort Queue using Header Sort 516
Store compile date/time in variables 422

5.1.3.1.2.1 TestTemplateQSort

This is a simple window that demonstrates the Add Header Sort to Queue 458 procedure extension template.

5.1.3.1.3 TestUtilityClass

Enter topic text here.

5.2 Video tutorials

Enter topic text here.

Part

Chapter 6 - File Attributes

6 File Attributes

The File Attributes are used by the Directory function.

```
ff_:NORMAL    EQUATE(0)       ! Normal files
ff_:READONLY  EQUATE(1)       ! Not for use as attributes parameter
ff_:HIDDEN    EQUATE(2)       ! Hidden files
ff_:SYSTEM    EQUATE(4)       ! System files
ff_:DIRECTORY EQUATE(10H)     ! Directories
ff_:ARCHIVE   EQUATE(20H)     ! NOT Win95 compatible
```

Part VII

Chapter 7 - API Reference

7 API Reference

This topic contains links to various sections of the MSDN website. We use it for our own reference and thought it might benefit others who are doing research into Win32 application apis on the MSDN website. Note that these links are created in June 2007.

Windows API Reference, Functions by category
Windows API Reference, Functions in alphabetical order

Index

- ? -

- @ -

- [-

- \ -

- __ -

- < -

- > -

- 1 -

- 2 -

- 3 -

- 4 -

- 6 -

- 9 -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- X -

- Y -

- Z -