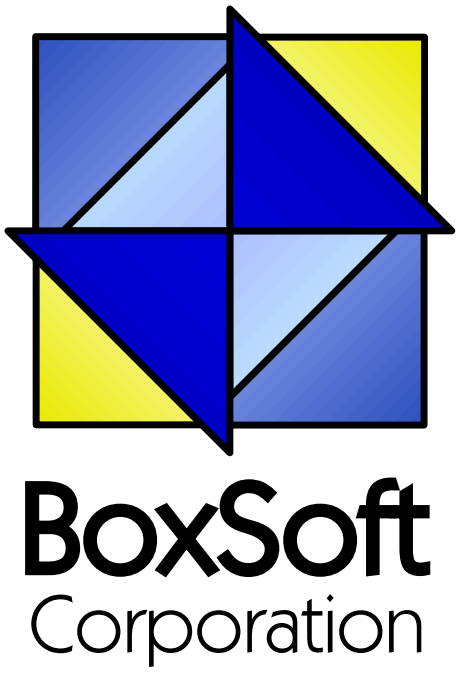


Super Field-Filler - Documentation

Table of Contents

Foreword 0

Part I Getting Started 3

1 Introduction ... 3
2 RTFM Warning!!! ... 4
3 Installation ... 5

Part II Template Usage 9

1 Adding a Field-Filler Procedure to your Applications ... 9
2 User-controlled Field Filler ... 10
3 Optionally Filter Records ... 14

Part III Appendices 15

1 Example Programs ... 15
2 What Are Tags? / Tag Storage Options ... 16
3 API Reference - Tagging ... 20
4 Project Defines ... 27
5 Troubleshooting ... 29
6 Contacting Technical Support ... 30
7 License Agreement ... 31

Index 32

1 Getting Started

1.1 Introduction

The BoxSoft Super Field-Filler templates enable the user to perform bulk ad hoc changes to a field across their entire file, without having to do it manually and without needed you to write a special procedure each situation.

This is extremely handy for those unexpected situations like a telephone area code change or a yearly status field modification. The most important thing is that it allows your program to be more flexible, without you having to predict all possible requirements.

The user can choose the field to be changed, the new value, the step value (for auto-numbering fields), and whether to process only tagged entries. You may remove any of these settings from the window, and you may also prevent some of the fields from being changed.

If you are using the SuperTagging features, you must have either of the BoxSoft SuperTagging or SuperQBE global support Extensions in place. Because the FieldFiller procedure template is basically a superset of the Process template, you can also use any regular filter expression.

ABC and Legacy Template Chains

This documentation pertains to both the ABC and Legacy (a.k.a. "Clarion") Super Template sets. In some situations we've implemented features in ABC that are not in Legacy, primarily because the old template chain was to be phased out. Due to customer pressures, however, Soft Velocity decided to reinstate support for the Legacy/Clarion chain.

Some of the Super Template features that are only in the ABC chain would be very difficult to implement in the legacy chain. However, we'll attempt to do this wherever it seems feasible to us. We apologize if this causes you any inconvenience. Please feel free to contact us if there's a particular feature in ABC that you would like to see in the Legacy chain, and we'll see if your needs can be accommodated.

For more information, see the following topics:

Adding Super Field-Filler to your Applications
- FieldFiller Procedure Template
- OptionalFilter Control Template

Example Programs
What Are Tags? / Tag Storage Options
API Reference
Project Defines
Troubleshooting
Contacting Technical Support
License Agreement

1.2 RTFM Warning!!!

It is very important that you read this documentation. If you follow the instructions step-by-step, then the usage is very simple. It is almost *impossible* if you try to do it on your own!

1.3 Installation

Installation Directory Structure

NOTE: As of version 6.6, we've changed our installation to the defacto "3rdParty" directory structure. (In Clarion 7 this is actually the "Accessory" directory.) Your old CLARIONx\SUPER directory has been renamed to CLARIONx\SUPER-OLD.

Once you've finished running the installation program, you should see the following structure under your C55 or Clarion6 directory:

```
C:\CLARION6, C:\C55, etc.
+-LibSrc        STA*.INC        (ABC headers)
+-3rdParty
| +-Bin ST_*.HLP, ST_*.CNT, STAB_CNV.DLL
| +-Template    STA?_*.TPL, STA*.TPW    (ABC chain)
| |     STC?_*.TPL, STC*.TPW    (Clarion chain)
| +-LibSrc      STA*.INC, STA*.CLW, STA*.TRN    (ABC chain)
| |     STC*.INC, STC*.CLW, STC*.TRN    (Clarion chain)
| +-Images
| | `-Super     *.ICO, *.CUR, *.WMF, *.GIF
| +-Docs
| | `-Super     *.PDF   (Documentation)
| `-Vendor
|   `-Super
|     +-QBE
|     | +-Examples
|     | | +-ABC *.DCT, *.APP, *.TPS (Examples)  (ABC chain)
|     | | `-Clarion     *.DCT, *.APP, *.TPS (Examples)  (Clarion chain)
|     | `-Source
|     |   +-ABC *.TXD, *.TXA, *.DCT (Source)    (ABC chain)
|     |   `-Clarion     *.TXD, *.TXA, *.DCT (Source)    (Clarion chain)
|     +-Tagging
|     | +-Examples
|     | | +-ABC *.DCT, *.APP, *.TPS (Examples)  (ABC chain)
|     | | `-Clarion     *.DCT, *.APP, *.TPS (Examples)  (Clarion chain)
|     | `-Source
|     |   +-ABC *.TXD, *.TXA, *.DCT (Source)    (ABC chain)
|     |   `-Clarion     *.TXD, *.TXA, *.DCT (Source)    (Clarion chain)
|     `-Etc.
|       +- . . .
`-SUPER-OLD             (ABC headers)
  `- . . .
```

For Clarion 7 and later versions it should look like this (note the two trees):

```
C:\Program Files\SoftVelocity\Clarion 7
`-Accessory
  +-Bin                 ST_*.HLP, ST_*.CNT, STAB_CNV.DLL
  +-Template
  | `-Win               STA?_*.TPL, STA*.TPW                    (ABC chain)
  |                     STC?_*.TPL, STC*.TPW                    (Clarion chain)
  |                     STMH*.TPW                               (Shared)
  +-LibSrc
  | `-Win               STA*.INC, STA*.CLW, STA*.TRN            (ABC chain)
  |                     STC*.INC, STC*.CLW, STC*.TRN            (Clarion chain)
  +-Images
  | `-Super             *.ICO, *.CUR, *.WMF, *.GIF
  `-Docs
    `-Super             *.PDF                                   (Documentation)
```

```
"My Documents" or "Shared Data" (depending on OS)
`-Clarion 7\Accessory
  `-Super
    +-QBE
    | +-Examples
    | | +-ABC           *.DCT, *.APP, *.TPS (Examples)          (ABC chain)
    | | `-Clarion       *.DCT, *.APP, *.TPS (Examples)          (Clarion chain)
    | `-Source
    |   +-ABC           *.TXD, *.TXA, *.DCT (Source)            (ABC chain)
    |   `-Clarion       *.TXD, *.TXA, *.DCT (Source)            (Clarion chain)
    +-Tagging
    | +-Examples
    | | +-ABC           *.DCT, *.APP, *.TPS (Examples)          (ABC chain)
    | | `-Clarion       *.DCT, *.APP, *.TPS (Examples)          (Clarion chain)
    | `-Source
    |   +-ABC           *.TXD, *.TXA, *.DCT (Source)            (ABC chain)
    |   `-Clarion       *.TXD, *.TXA, *.DCT (Source)            (Clarion chain)
    `-Etc.
      +- . . .
```

To prevent conflicts between old Super Template files and same-named files in our new directory structure, the new installers attempt to delete the old files. If it encounters problems, then an error will be reported during the installation. Then you must delete any of the following files from the old directories, if they also exist in the new directory structure:

```
C:\CLARION6, C:\C55, etc.
+-LibSrc                STAB*.CLW, STCL*.CLW, STAM*.CLW, STCM*.CLW,
|                       STAB*.TRN, STCL*.TRN, STAM*.TRN, STCM*.TRN,
|                       STDEBUG.*
+-Template              STAB*.TP?, STCL*.TP?, STAM*.TPW, STCM*.TPW,
|                       STGROUPS.TPW, STDEBUG.TPW
`-Bin                   ST_*.HLP, ST_*.CNT, ST_*.GID
```

For example, you can use a tool like the indispensable Beyond Compare (www.scootersoftware.com) to investigate the contents of C:\Clarion6\LibSrc and C:\Clarion6 \3rdParty\LibSrc. View only files matching the mask ST*.* and hide all "orphans", which will show the files that exist in both directories. Delete the files from C:\Clarion6\LibSrc, and then do the same for the Template and Bin directories.

Filenames and Product Abbreviations

- Super Template filenames generally start with the letters "ST". That's about all you can go on most of the time. (Our image files don't follow this convention, but they are sequestered in the Image\Super subdirectory.
- The next two letters are usually AB (for the ABC chain) or CL (for the Clarion/legacy chain). One exception is Super Stuff (MH), which uses AM, CM and MH. Also, if both the ABC and Clarion chain share a TPW, then the AB/CL are skipped and it goes on to the product abbreviation. (Again, Super Stuff is an exception, as it uses MH for the shared files.)
- There are several TPWs that are shared by multiple Super Templates: STGROUPS.TPW, STABABC.TPW, STBLDEXP.TPW, STDEBUG.TPW
- The last four characters:
  - For TPL files, the last four letters are an underscore, followed by one of the following

suffices. The exceptions are STABAEQB.TPL and ST?M_STF.TPL.

- For TPW files, the last four letters may match one of these in its entirety, or be followed by additional characters denoting the special purpose files.
- Super Stuff (MH) is an exception, in that it uses STcMxxxx, where "c" is the chain of A or C, and "xxxx" denotes the special purpose.

AEQB	Super QuickBooks-Export (i.e. Accounting-Export QuickBooks)
BRW/BW	Super Browse
DIA	Super Dialer
FF	Super Field-Filler
IE	Super Import-Export
INV	Super Invoice
LIM	Super Limiter
PCD	Super Passcode
QBE	Super QBE
SEC	Super Security
TAG	Super Tagging
MH/STF	Super Stuff (MH) *(a.k.a. The "MikeHanson" Templates)*

Update the Redirection File

The installation program is able to update your redirection file automatically. If you decline the option during the installation, then you will have to edit the redirection file yourself. The three things that must be found are the Templates, LibSrc and Images. For example, you might make the following changes to the the *.* entry in Clarion 6:

```
*.*   = .; %ROOT%\examples; %ROOT%\libsrc; %ROOT%\images; %ROOT%\template; %ROOT%
              \3rdParty\template; %ROOT%\3rdParty\libsrc; %ROOT%\3rdParty\images\super
```

In Clarion 7 and above it will be more like this:

```
*.*   = %ROOT%\Accessory\images; %ROOT%\Accessory\resources; %ROOT%\Accessory\libsrc\win; %
              ROOT%\Accessory\template\win; %ROOT%\Accessory\images\Super
```

There are *.RED examples in the SUPER\DOC directory.

Register the Templates

Clarion allows you to have multiple template sets accessible in the same application. It does this with the Template Registry. To use a Super Template, you must register it first. The installation program attempts to do this for you, but in case it fails, or if your registry becomes corrupted, then you must register them manually.

1. Load Clarion, then select the "Setup / Template Registry" pulldown menu option.
2. Press the [Register] button.
3. Select C:\CLARION\3rdParty\Template\ST_*.TPL (ABC) or ST_*.TPL (Clarion). The directory name may not exactly match your system.

Assuming this all went without a hitch, you're ready to start using the templates.

2 Template Usage

2.1 Adding a Field-Filler Procedure to your Applications

There are two templates included with the Super Field-Filler. The first is the FieldFiller Procedure template providing the basic Window (including the Fields list, New Value and Step Value fields).

The second is the OptionalFilter Control template that lets your users specify whether to process only tagged records. Alternatively, you can provide your own filter expression.

For more information, see the following topics:

FieldFiller Procedure Template
OptionalFilter Control Template

2.2 User-controlled Field Filler

(FieldFiller Procedure Template)

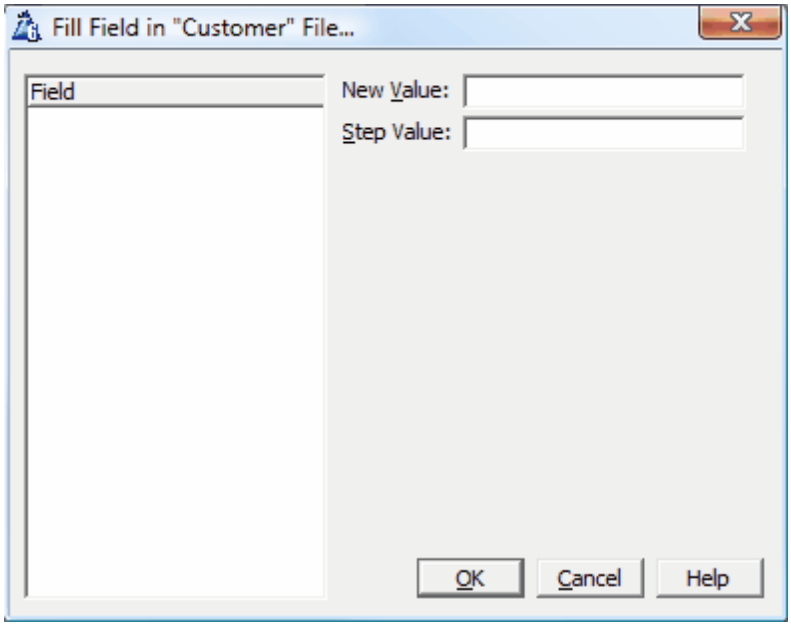
This procedure template allows your users to change the value of a field in all of the records of the data file. (A regular filter expression can be used to restrict these, or you can use the included OptionalFilter control template.) The user specifies the new value for a field, and an optional "step" value for auto-numbering.

The user is presented a list of fields available for modification. The developer controls which fields appear in this list using a combination of dictionary defaults, and explicit inclusions and exclusions.

The FieldFiller automatically uses the picture from the dictionary definition of the field. You can override this picture, or even supply an alternate "override control" for entering the value.

This procedure template is actually a superset of Clarion's Process procedure template. All of the same settings and embeds are available. This means that you can specify a series of related files under your primary file to create a complex view of your system (e.g.: Customer-Invoice-InvItem-Product-Supplier). The main difference is that there is one extra object: the WindowManager is regular one (like a Form), while the ExportManager plays the part of the ProcessManager. The ExportManager is not called until the WindowManager tells it to proceed.

The default window contains the following controls:



Field List Box - This list box contains a separate entry for each field found in the file. The user specifies which field to change by highlighting it here. You should not remove this list box.

"New Value" Field - This is used to specify the new value for the field. Depending on the selected field, the entry picture will change appropriately. If you specify override

controls, then they will be overlaid on this position. You should not remove this control.

"Step Value" Field - This is used to enter the optional step value for auto-numbering the field. (In this case, the "New Value" will be treated as the starting point.) You can selectively hide the Step control for each field. You can also remove the Step control entirely.

[OK] Button - This begins the FieldFiller process. You should not remove this control.

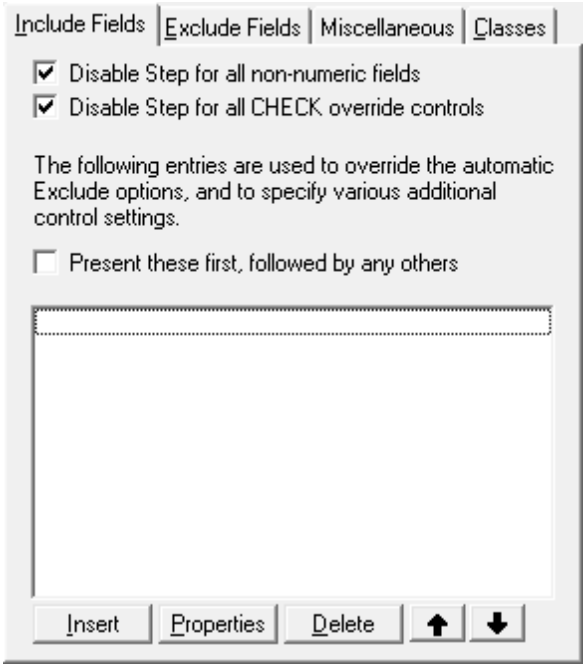
[Cancel] Button - This cancels the FieldFiller before it is started. You should not remove this control.

[Help] Button - This calls the help window. You must set-up the help topics yourself. If desired, you may remove this control.

Optional Controls - If you wish to override the "New Value" control for a specific field (for CHECKs, DROPLISTs, etc.), then simply add the field's control to the window. (Do not include the prompt.) The template will automatically hide the control when the window is opened, then overlay it on the New Value control when needed. For an demonstration of this, see the accompanying example program.

The rest of the settings are behind the [FieldFiller Properties] button on the main Procedure Properties window. They are as follows:

Include Fields Tab



Disable Step for all non-numeric fields - This setting causes the "Step Value" field to be hidden for all non-numeric fields, including STRING, CSTRING, PSTRING, MEMO, etc.

Disable Step for all CHECK override controls - This setting causes the "Step Value" field to be

hidden when using "override fields" of the CHECK type.

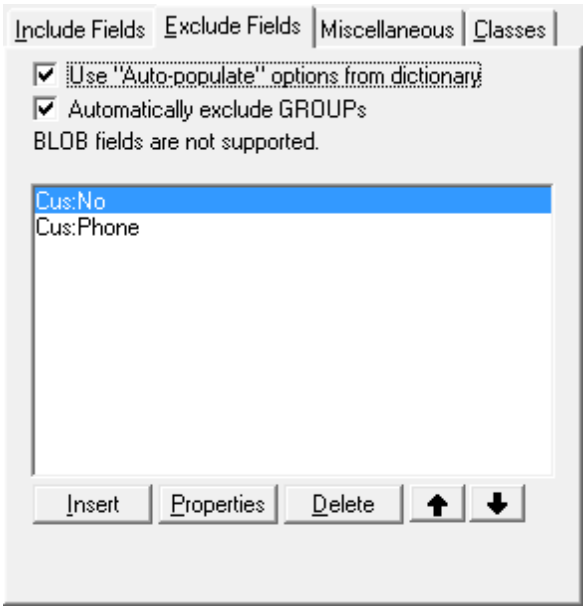
Individual Fields - Here you can specify various field specific settings. If you wish to override your settings for "dictionary auto-populate" and "exclude groups" on the "Exclude" Tab, then add your field to the list. The template automatically includes all fields, so this is necessary only as an override against automatic exclusions.

If you want to specify a dimensioned field, it is treated like a group in the pop-up window. You must expand its tree before selected the desired array element.

For each field, you may specify an alternative entry picture (only used in the absence of an override control). You can also indicate that the Step Value should be hidden.

Present these first, followed by any others - This forces the fields added to the list to appear at the top of the Field-Filler list, followed by any automatic additions. The additional benefit of this is that you can use it to control the order in which fields appear. Usually they will appear in the order in which they are defined in the file. If you add all the fields to this list, then you can tweak the order as needed.

Exclude Fields Tab



There is no referential integrity applied during FieldFiller updates. Therefore, you should make prudent use of the Exclude feature, so that your users do not change fields that are crucial to the integrity of your data relationships.

Use "Auto-populate" options from dictionary - This tells the template to check your dictionary to determine which fields the user should not see. Individual fields can be overridden on the "Include" tab.

Automatically exclude GROUPs - This tells the template to exclude any GROUP fields that it finds in your file. Individual groups can be overridden on the "Include Fields" tab.

Individual Fields - This list box contains any additional fields that you wish to exclude. If you want to specify a dimensioned field, it is treated like a group in the pop-up window. You must expand its tree before selected the desired array element.

Miscellaneous Tab

Use Description from DCT instead of field name - If your field names are not easily understood by the user, this switch will cause FieldFiller to use the descriptions instead (where available).

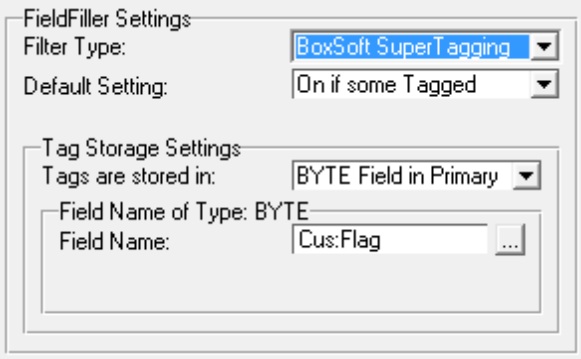
Classes Tab

This contains the standard ABC class settings tab.

2.3 Optionally Filter Records

(OptionalFilter Control Template)

This template allows your users to decide at run-time whether the developer-specified filter should be used to control which records are modified. After you populate the control on the window, you can alter the following extension settings:



Filter Type - This will be either "BoxSoft SuperTagging" or "Filter Expression". If you choose SuperTagging, the "Tag Storage" tab will be present. If you wish to specify your own filter, then select Filter Expression and enter the expression below.

Default Setting - This determines the initial setting of the control. The options are "On", "Off", or "On if some tagged". The first two are self-explanatory; the third indicates that it should be On only if some records match the filter.

Tag Storage Settings - This will appear only if you chose "BoxSoft SuperTagging" for the "Filter Type" in the General tab. These settings specify the location of the tags, and they must match the settings where the tags were originally created.

Filter Expression - This will appear only if you chose "BoxSoft SuperTagging" for the "Filter Type" in the General tab. This is the optional filter expression. If you wanted to use the FieldFiller to change only new customers, then you could type Cus:New=True here, and change the control's display text on the window accordingly.

3 Appendices

3.1 Example Programs

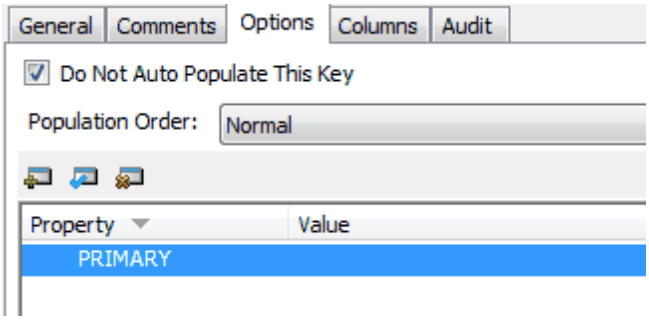
There are two examples included: one for the ABC template chain and one for the Clarion legacy chain. Each demonstrates the FieldFiller in four procedures: using all options, without the filter, without the step, and without both. In all procedures, override fields are used for the CHECK and DROPLIST fields.

3.2 What Are Tags? / Tag Storage Options

You will often be asked to specify your Tag Storage options. For a particular file, you should be as consistent as possible when entering these. Of course, there will be exceptions when you normally store tags in memory, but you want to save them to disk.

NOTE: For the Memory and Disk Set tag storage methods, the tags are stored separately from your data file, with a pointer back to your record's primary key field. This means that your data file must contain a key marked as Unique and Primary. The key must contain only one field component. Either an integer (e.g.: LONG) or a string will do. Of course, a LONG is preferable, in which case the key should also be Auto-Numbered.

If you *cannot* mark one of your keys as Primary, then a suitable Unique key can be used instead. To tell Super Tagging to use the key, add the following setting to the key's options in the dictionary:



What is a "Tag Set"

Think of it as a collection of tags associated with a particular file and used for a certain purpose. You should use a different Tag Set number for each data file; otherwise, there is a very good chance that your tag sets will corrupt each other. If you need two different types of tags associated with the same file, just use two different tag set numbers.

This can be a constant (e.g. **1**), an equate (e.g. **eT_Customer or Tag:Customer**), or a variable (e.g. **Loc:TodaysTags**). It's a good idea to define tag sets as global equates, rather than integer constants. For example:

```
            ITEMIZE(1),PRE(Tag)
Customer      EQUATE
Employee      EQUATE
Product       EQUATE
Invoice       EQUATE
EmpInvoice    EQUATE  ! separate invoice tag set
            END!ITEMIZE
```

Tag Storage Settings in Clarion/Legacy versus ABC

In the ABC templates, you have the option of:

- Memory Set

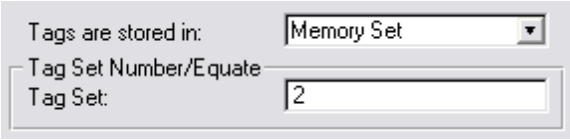
- Disk Set
- BYTE Field in Primary

The legacy templates differ, in that they offer:

- Memory (POINTER)
- Memory (POSITION)
- TagFile_ (POINTER) i.e. "Disk"
- TagFilePos_ (POSITION) i.e. "Disk"
- BYTE Field in Primary

The key difference is that the ABC templates automatically determine whether your primary key is an integer (e.g. LONG). If it's an integer, it uses the "POINTER" method, which are really just whole number tag references. Otherwise, it uses the "POSITION" method, which stores the tag reference as STRINGs.

Memory Set



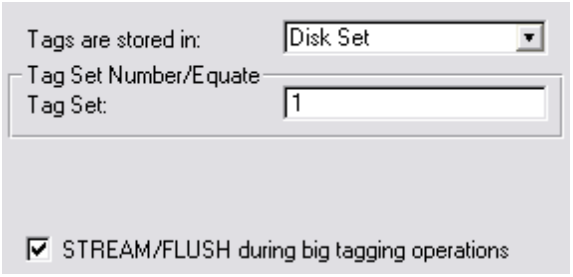
Using this tag storage method, the tags are stored in a memory queue. Each entry contains a LONG integer that equals your data record's primary key field.

Because the tags are stored in memory, multiple users can tag records without interfering with each other. There are two disadvantages to this method:

1. The tags are only static while the program is running. If you need to save the tags from one session to the next, you can use the SaveTags or CopyTags templates, or you can use "Disk Set" instead.

2. If you are tagging only a few records of a large data file, and you wish to process only the tagged records, using Memory Set forces you to use a slow filter rather than a fast range. Again, you could copy the tags to a "Disk Set" before running the Process or Report, but this can get a little confusing.

Disk Set



The difference between this and "Memory Set" is that the tags are stored on disk. It's slower, but

the tags are remembered from session to session. You can also use it directly for Reports and Processes, looking-up the related data record as each tag is processed.

NOTE: If you wish to process tagged records in ReportWriter, you will need to save them on disk.

Tags are stored in either TagFile_ or TagFilePos_, depending on whether your primary key field is a numeric or string field. TagFile_ and TagFilePos_ are separate files usually using the TopSpeed file driver (or other driver of your choice).

Each record contains the user number, tag set number, primary key, and sorting value. This enables you to have multiple users tagging records in the file at the same time. Or a single user could create multiple tag sets for the same file (e.g.: one for billing and another for advertising).

The main disadvantage to this method is the slowness with which tags are added and deleted. However, if you need to report on a small number of tagged records from a large data file, this is the preferable method.

STREAM/FLUSH during big tagging operations (not always present) - If only one person is normally using the system, then you may want to turn this on. It locks the TagFile during these operations, so you shouldn't use the feature in a multi-user environment unless you don't have many users tagging at the same time.

BYTE Field in Primary

This tag storage method toggles a BYTE field in your primary file. It is not multi-user compatible, but this may not be an issue for you. It happens to be the easiest to use in Reports and Processes (especially ReportWriter). It is not, however, the most efficient. In some ways it is faster than using Disk Set, and in some ways slower.

You may wish to use one of the other tagging methods for the actually tagging itself, then use the CopyTags code template to save the tags in the BYTE field before running a Report or Process.

Field Name - This is the field that will hold the "Tagged" status.

Tagged Value *(not always present)* - This is the value assigned to your "BYTE Field in Primary" when the record is tagged. It applies only if you are using that storage method. You can specify a number, string or variable name (without preceding exclamation). The value is used without modification (e.g.: Pre:TagField = YourSetting). The default is "True".

STREAM/FLUSH during big tagging operations *(not always present)* - If only one person is normally using the system, then you may want to turn this on. It locks the primary data file during these operations, so you shouldn't use the feature in a multi-user environment.

3.3 API Reference - Tagging

Tagging API for Memory (Integer Primary Keys)

If you are using the "Memory Set" storage method and some of your data files have integer primary key fields, then the following procedures and functions apply:

ClrTag:PtrM (TagSet, PrimaryKey)

This procedure is used to clear a tag for a particular tag set and primary key.

```
ClrTag:PtrM(1, Cus:No)
```

CntTag:PtrM (TagSet)

This function returns the number of tags in the specified tag set.

FstTag:PtrM (TagSet)

This function returns a pointer to the first tagged record in the specified tag set. If there are no records in the specified tag set, the function returns zero.

There are a number of situations where you would use FstTag:PtrM(). You may want to check if there are any tagged records available. You would also use FstTag:PtrM() in conjunction with NxtTag:PtrM() to process all tagged records.

GetTag:PtrM (TagSet, PrimaryKey)

This function returns either 1 or 0, depending on whether the requested record is tagged.

```
IF GetTag:PtrM(1, Cst:No)
  DO Process
END
```

ListTags:PtrM (TagSet)

This function returns a string with a list of tags as '[1][2][3]...'. This can be useful for filters with INSTRING functions, etc. The maximum string length is 1000 characters. If your tag set exceeds that, an ASSERT will fail.

LstTag:PtrM (TagSet)

This function returns a pointer to the last tagged record in the specified tag set. If there are no records in the specified tag set, the function returns zero.

There are a number of situations where you would use LstTag:PtrM(). You may want to check if there are any tagged records available. You would also use LstTag:PtrM() in conjunction with PrvTag:PtrM() to process all tagged records.

NewTag:PtrM (TagSet)

This procedure clears all tags for a specified tag set.

NxtTag:PtrM (TagSet)

This function returns a pointer to the next tagged record in the specified tag set. FstTag:PtrM() must be called first for this function to work. If there are no more records, the function returns zero.

PrvTag:PtrM (TagSet)

This function returns a pointer to the previous tagged record in the specified tag set. LstTag:PtrM () must be called first for this function to work. If there are no more records, the function returns zero.

RvsTag:PtrM (TagSet, PrimaryKey)

This procedure is used to reverse a tag for a particular tag set and primary key. That is, if the tag is set before the call to RvsTag:PtrM it will be cleared, and vice versa. There is an optional Value parameter, for storing the tags in a particular order for FstTag:PtrM / NxtTag:PtrM processing.

```
RvsTag:PtrM(1, Prd:No)
RvsTag:PtrM(2, Cus:No)
```

SetTag:PtrM (TagSet, PrimaryKey)

This procedure is used to set a tag for a particular tag set and primary key. There is an optional Value parameter, for storing the tags in a particular order for FstTag_/NxtTag_ processing. It returns True if newly tagged, or False if already tagged.

```
SetTag:PtrM(1, Prd:No)
SetTag:PtrM(2, Cus:No)
```

ShutDownTag:PtrM ( )

This procedure clears all of the tag queues in memory before the program exits.

Tagging API for Memory (Non-Integer Primary Keys)

If you are using the "Memory Set" storage method and some of your data files have non-integer primary key fields, then the following procedures and functions apply:

ClrTag:PosM (TagSet, PrimaryKey)

This procedure is used to clear a tag for a particular tag set and primary key.

```
ClrTag:PosM(1, Cus:ID)
```

CntTag:PosM (TagSet)

This function returns the number of tags in the specified tag set.

FstTag:PosM (TagSet)

This function returns a string pointing to the first tagged record in the specified tag set. If there are no records in the specified tag set, the function returns an empty string ('').

There are a number of situations where you would use FstTag:PosM(). You may want to check if there are any tagged records available. You would also use FstTag:PosM() in conjunction with NxtTag:PosM() to process all tagged records.

GetTag:PosM (TagSet, PrimaryKey)

This function returns either 1 or 0, depending on whether the requested record is tagged.

```
IF GetTag:PosM(1, Cus:ID)
  DO Process
END
```

ListTags:PosM (TagSet)

This function returns a string with a list of tags as '[1][2][3]...'. This can be useful for filters with INSTRING functions, etc. The maximum string length is 1000 characters. If your tag set exceeds that, an ASSERT will fail.

LstTag:PosM (TagSet)

This function returns a string pointing to the last tagged record in the specified tag set. If there are no records in the specified tag set, the function returns an empty string ('').

There are a number of situations where you would use LstTag:PosM(). You may want to check if there are any tagged records available. You would also use LstTag:PosM() in conjunction with PrvTag:PosM() to process all tagged records

NewTag:PosM (TagSet)

This procedure clears all tags for a specified tag set.

NxtTag:PosM (TagSet)

This function returns a string pointing to the next tagged record in the specified tag set. FstTag: PosM() must be called first for this function to work. If there are no more records, the function returns an empty string ('').

PrvTag:PosM (TagSet)

This function returns a string pointing to the next tagged record in the specified tag set. LstTag: PosM() must be called first for this function to work. If there are no more records, the function returns an empty string ('').

RvsTag:PosM (TagSet, PrimaryKey)

This procedure is used to reverse a tag for a particular tag set and primary key. That is, if the tag is set before the call to RvsTag:PosM it will be cleared, and vice versa. There is an optional Value parameter, for storing the tags in a particular order for FstTag:PosM / NxtTag:PosM processing.

```
RvsTag:PosM(1, Prd:ID)
RvsTag:PosM(2, Cus:ID)
```

SetTag:PosM (TagSet, PrimaryKey)

This procedure is used to set a tag for a particular tag set and primary key. There is an optional Value parameter, for storing the tags in a particular order for FstTagPos_/NxtTagPos_ processing. It returns True if newly tagged, or False if already tagged.

```
SetTag:PosM(1, Prd:ID)
SetTag:PosM(2, Cus:ID)
```

ShutDownTag:PosM ( )

This procedure clears all of the tag queues in memory before the program exits.

Tagging API for TagFile_ (Integer Primary Keys)

If you are using the "Disk Set" storage method and some of your data files have integer primary key fields, then the following procedures and functions apply:

ClrTag_ (TagSet, PrimaryKey, <Value>)

This procedure is used to clear a tag for a particular tag set and primary key.

```
ClrTag_(1, Cus:No)
```

CntTag_(TagSet)

This function returns the number of tags in the specified tag set.

FstTag_ (TagSet)

This function returns a pointer to the first tagged record in the specified tag set. If there are no records in the specified tag set, the function returns zero. It processes the tags in TF_:ValKey order, which is usually the order of the original browse or order that the records were tagged.

There are a number of situations where you would use FstTag_(). You may want to check if there are any tagged records available. You would also use FstTag_() in conjunction with NxtTag_() to process all tagged records.

GetTag_(TagSet, PrimaryKey)

This function returns either 1 or 0, depending on whether the requested record is tagged.

```
IF GetTag_(1, Cus:No)
  DO Process
END
```

ListTags_ (TagSet)

This function returns a string with a list of tags as '[1][2][3]...'. This can be useful for filters with INSTRING functions, etc. The maximum string length is 1000 characters. If your tag set exceeds that, an ASSERT will fail.

LstTag_ (TagSet)

This function returns a pointer to the last tagged record in the specified tag set. If there are no records in the specified tag set, the function returns zero. It processes the tags in reverse TF_: ValKey order

There are a number of situations where you would use LstTag_(). You may want to check if there are any tagged records available. You would also use LstTag_() in conjunction with PrvTag_() to process all tagged records.

NewTag_ (TagSet, <Stream>)

This procedure clears all tags for a specified tag set.

NxtTag_ (TagSet)

This function returns a pointer to the next tagged record in the specified tag set. FstTag_() must be called first for this function to work. If there are no more records, the function returns zero.

PrvTag_ (TagSet)

This function returns a pointer to the previous tagged record in the specified tag set. LstTag_() must be called first for this function to work. If there are no more records, the function returns zero.

RvsTag_ (TagSet, PrimaryKey, <Value>)

This procedure is used to reverse a tag for a particular tag set and primary key. That is, if the tag is set before the call to RvsTag_ it will be cleared, and vice versa. There is an optional Value parameter, for storing the tags in a particular order for FstTag_/NxtTag_ processing.

```
RvsTag_(1, Prd:No)
RvsTag_(2, Cus:No)
```

SetTag_ (TagSet, PrimaryKey, <Value>)

This procedure is used to set a tag for a particular tag set and primary key. There is an optional Value parameter, for storing the tags in a particular order for FstTag_/NxtTag_ processing. It returns True if newly tagged, or False if already tagged.

```
SetTag_(1, Prd:No)
SetTag_(2, Cus:No)
```

Tagging API for TagFilePos_ (Non-Integer Primary Keys)

If you are using the "Disk Set" storage method and some of your data files have non-integer primary key fields, then the following procedures and functions apply:

ClrTagPos_ (TagSet, PrimaryKey, <Value>)

This procedure is used to clear a tag for a particular tag set and primary key.

```
ClrTagPos_(1, Cus:ID)
ClrTagPos_(1, Cst:ID)
```

CntTagPos_ (TagSet)

This function returns the number of tags in the specified tag set.

FstTagPos_ (TagSet)

This function returns a string pointing to the first tagged record in the specified tag set. If there are no records in the specified tag set, the function returns an empty string (''). It processes the tags in TP_:ValKey order, which is usually the order of the original browse or order that the records were tagged.

There are a number of situations where you would use FstTagPos_(). You may want to check if

there are any tagged records available. You would also use FstTagPos_() in conjunction with NxtTagPos_() to process all tagged records.

GetTagPos_ (TagSet, PrimaryKey)

This function returns either 1 or 0, depending on whether the requested record is tagged.

```
IF GetTagPos_(1, Cus:ID)
  DO Process
END
```

ListTagsPos_ (TagSet)

This function returns a string with a list of tags as '[1][2][3]...'. This can be useful for filters with INSTRING functions, etc. The maximum string length is 1000 characters. If your tag set exceeds that, an ASSERT will fail.

LstTagPos_ (TagSet)

This function returns a string pointing to the last tagged record in the specified tag set. If there are no records in the specified tag set, the function returns an empty string (''). It processes the tags in reverse TP_:ValKey order

There are a number of situations where you would use LstTagPos_(). You may want to check if there are any tagged records available. You would also use LstTagPos_() in conjunction with PrvTagPos_() to process all tagged records

NewTagPos_ (TagSet, <Stream>)

This procedure clears all tags for a specified tag set.

NxtTagPos_ (TagSet)

This function returns a string pointing to the next tagged record in the specified tag set. FstTagPos_() must be called first for this function to work. If there are no more records, the function returns an empty string ('').

PrvTagPos_ (TagSet)

This function returns a string pointing to the next tagged record in the specified tag set. LstTagPos_() must be called first for this function to work. If there are no more records, the function returns an empty string ('').

RvsTagPos_ (TagSet, PrimaryKey, <Value>)

This procedure is used to reverse a tag for a particular tag set and primary key. That is, if the tag is set before the call to RvsTagPos_ it will be cleared, and vice versa. There is an optional Value parameter, for storing the tags in a particular order for FstTagPos_/NxtTagPos_ processing.

```
RvsTagPos_(1, Prd:ID)
RvsTagPos_(2, Cus:ID)
```

SetTagPos_ (TagSet, PrimaryKey, <Value>)

This procedure is used to set a tag for a particular tag set and primary key. There is an optional

Value parameter, for storing the tags in a particular order for FstTagPos_/NxtTagPos_ processing. It returns True if newly tagged, or False if already tagged.

```
SetTagPos_(1, Prd:ID)
SetTagPos_(2, Cus:ID)
```

Tagging API Miscellaneous

ST::GetTagUsr ( )

This function returns the value of the variable passed into UsrTag_(), or zero if UsrTag_() has not yet been called.

ST::SelectTagSet (Task, Source)

This procedure presents a browse of tag sets for the SaveTags and LoadTags control templates. The user can insert, change, delete and select tag sets. As with other Browse procedure, it sets GlobalResponse appropriately.

The "Task" parameter is either 'S' for Save or 'L' for Load. The "Source" parameter indicates the origin of the tags. This will contain the label of the data file (e.g.: Customer, Employee)

If you don't like the way this window looks, you can turn off this procedure in the global extension options and create your own using the TagSet_ file.

You can import SUPER\LIBSRC\TAGGING\TAGGING.TXA as a starting point. The two procedures use the regular Clarion Browse and Form procedure templates. Make sure that you have the Tag files in your dictionary before you import the TXA file. (Import SUPER\LIBSRC\TAGGING\TAGGING.TXD into your dictionary.)

UsrTag_ (UserNo_)

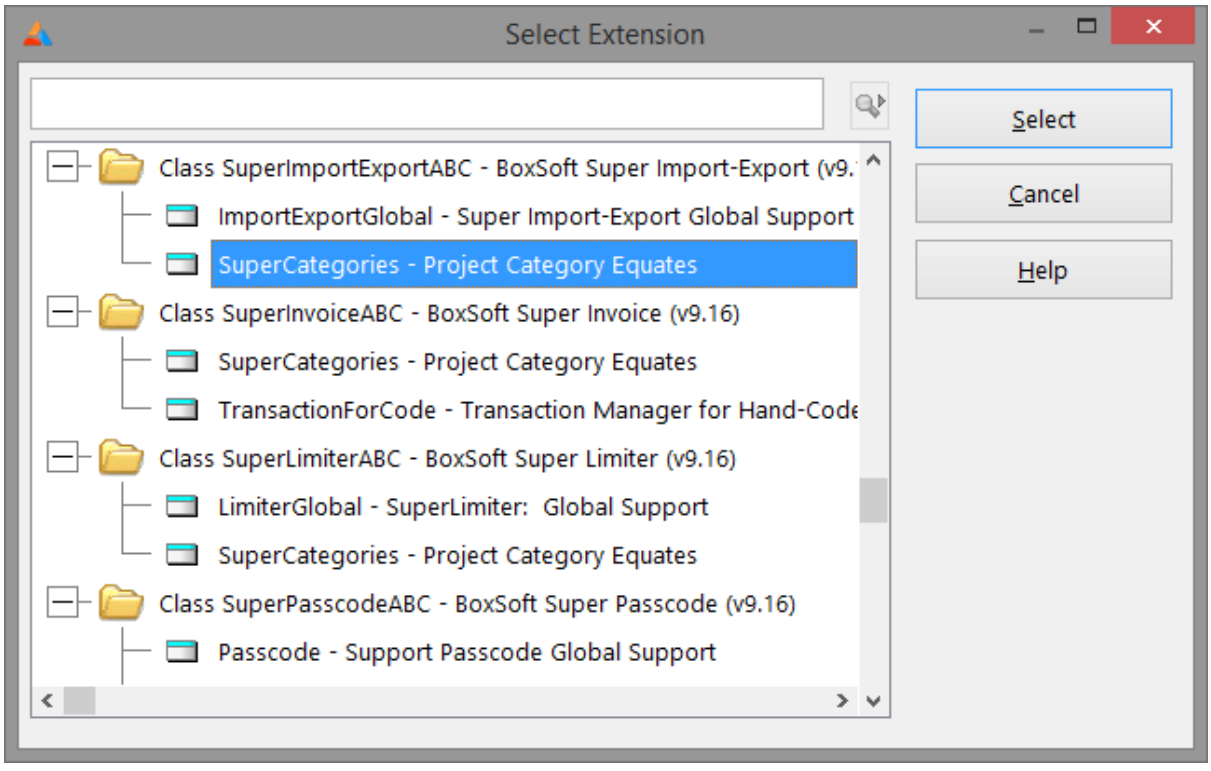
This procedure is used to set the value of the current "user". If you're using the BoxSoft Super Security templates, then this is handled for you automatically. If you're using another method for security, you need to call this procedure whenever your user changes. Usually this will be only at the start of the program.

3.4 Project Defines

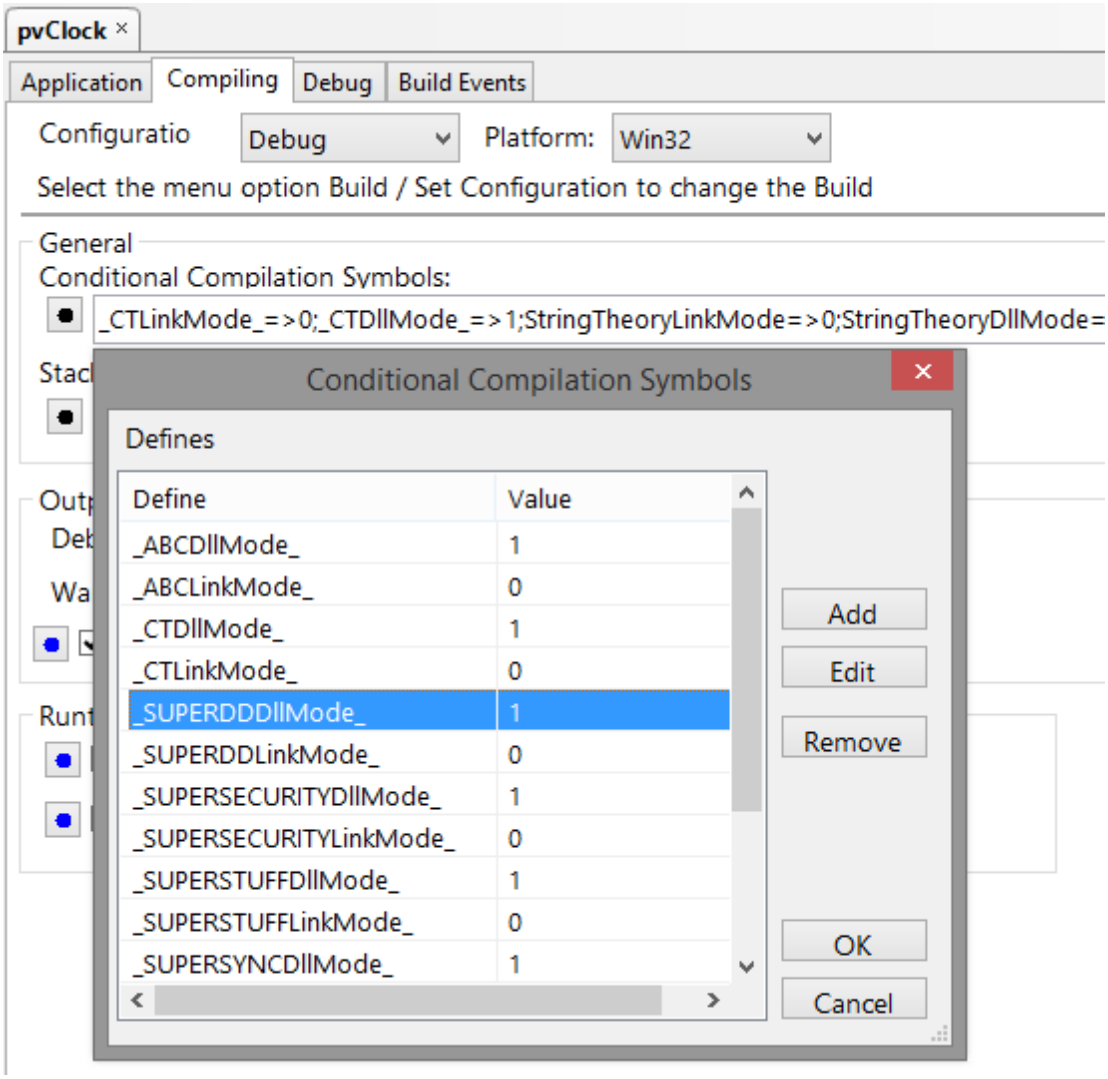
Prior to our 7.0 template versions, we were utilizing the same LINK and DLL Project Defines as the ABC libraries: _ABCLinkMode_ and _ABCDllMode_. This caused all of our libraries to be included in your base/dictionary DLL, even if you weren't using them in a particular development project.

To make this inclusion more selective, as of our 7.0 versions we changed to use various other switches. Some of these are product related (e.g. Super QBE uses _SuperQBELinkMode_ and _SuperQBEDllMode_), while others are associated with one of our shared base classes (e.g. Drag & Drop uses _SuperDDLinkMode_ and _SuperDDDllMode_). Usually the templates (especially the global ones) automatically add the necessary entries to your Project Defines. If you happen to use the templates in your APPs in the wrong combination, these can be inadvertently omitted.

For APP-based systems, you can force the switches to be included by using the Super Categories global extension template. Every one of the Super Templates has this extension to apply its own switches, so if you're using multiple templates in a particular APP, you may have to add this extension for each of the products. (As was mentioned above, if there's already a global extension populated for a given Super Template, then you don't have to add this extension for that product.) Even if it's not needed, there's no problem with adding the SuperCategories global extension.



For hand-coded PRJ-based systems, you must add the switches manually. Take note of their names in the INC files, and then add them to the project settings like this:



3.5 Troubleshooting

There are no common troubleshooting problems to mention at this time.

3.6 Contacting Technical Support

If you have any troubles with this product, then please contact:

Mitten Software
2354 West Wayzata Blvd
Second Floor, Suite H
Long Lake, MN 55356

Voice: (952) 745-4941
Fax: (952) 745-4944

Internet: www.mittensoftware.com
answers@mittensoftware.com
www.boxsoft.net
www.boxsoft.net/contact.htm

3.7 License Agreement

One License per Developer

This Super Template product is comprised of the templates, default applications, libraries, source code, documentation, and help files provided with the package. You must have a separate registered copy for each developer using it.

Redistribution

You are allowed to use the product for any programs that you create, and you are permitted to distribute the generated source code. You may not, however, distribute any portion of the product in its original or modified form without the prior written consent from BoxSoft Development.

One exception to this is the example programs provided with this installation or separately from BoxSoft or its agents: these may be distributed without penalty, in either their original or a modified state.

Disclaimer

BoxSoft Development does not warranty this software for any use. Any expenses or lost time due to errors in this product are not the responsibility of BoxSoft Development. We will attempt to fix any errors that are brought to our attention, but we are not legally liable for any lack of correctness of the product.

Index

- A -

Adding a Field Filler Procedure to your Applications 9
API Reference 20

- B -

BYTE Field in Primary 16

- C -

Class Libraries 5, 27
Contacting Technical Support 30

- D -

Defines 27
Directories 5
Disappear 27
Disk Set 16
DLLMode 27

- E -

Examples 15

- F -

Field Filler Procedure 10
Freeze 27
Function Reference 20

- G -

GPF 27

- I -

Include Files 5
Installation 5

- L -

Library Reference 20
License Agreement 31
LinkMode 27

- M -

Memory Set 16

- P -

Procedure Reference 20
Project Defines 27

- R -

Redirection File 5
Registering the Template 5
RTFM Warning 4

- S -

Support 30

- T -

Tag Storage Options 16
Tagging 20
Tech Support 30
Template Registry 5
Troubleshooting 29

- U -

User-controlled Field Filler 10

- V -

Variable Reference 20