Super Tagging - Documentation

Table of Contents

1 Getting Started

1.1 Introduction

The Super Tagging Templates enable you to perform tagging operations using Clarion. Tagging can be very useful in many situations. All of these, however, can be generalized into a single purpose: to permit the user to randomly select one or more records for inclusion in a Report or Process.

You accomplish this by using a combination of templates and/or API procedures. You can use the templates and entirely ignore the API, or you can selective call those procedures when you have to handle an unusual situation. However, you'll be surprised at how much Super Tagging can do using only the templates.

You also have a choice as to where you want to store your tags. This enables you to handle any combination of multi-user, multi-table tagging operations. You'll read more about this later.

For now, lets take a quick look at some of the key features of Super Tagging:

BrowseTaggingButtons - This Control template lets you perform tagging operations using a regular Clarion BrowseBox. There are buttons to tag, untag, flip (toggle), tag all, untag all, flip all, tag rest (from current position), and untag rest. There are two additional buttons to jump to the next tag and previous tag. You can also click in the tag column to flip the tag.

BrowseOptTagFilter - This Control template places a button on your window which will show "Tagged ", "Untagged" or "All" records. Then the user can control which set of records they wish to see by pressing this button.

BrowseTagFilter - This Extension template enables you to display only tagged records in a browse. This differs from the previous feature in that the filter is always "on".

ProcessTagFilter - This Extension template adds a filter to a Report or Process so that only tagged records are used by the procedure. This is similar to the BrowseTagFilter template, except that it is specifically used for Report and Process procedures, rather than BrowseBox windows. For a faster solution, see the "ProcessGetTaggedData" template.

BrowseMarking - This Extension template provides Windows Standard Behavior (WSB) marking in a regular Clarion BrowseBox. Your users can press Click (MouseLeft) to mark a single record, Shift-Click (ShiftMouseLeft) to mark a range, and Ctrl-Click (CtrlMouseLeft) to toggle individual records.

ProcessGetTaggedData - This Extension template provides a faster alternative to the "ProcessTagFilter" template. With it, you process your report directly against TagFile_ or TagFilePos_, and this template will look up the tagged record. (If you are using your file's primary key field as the reference value stored by the tagging system, then you can use normal file relationships to perform this function instead of using this template.)

SaveTags - This Control template saves the current set of tags from the BrowseBox into a separate tag set that can be retrieved later with the "LoadTags" control template.

LoadTags - This Control template retrieves the tags that were saved earlier with the "SaveTags" control template.

CopyTags - This Code template copies tags from one tag set to another. If the "SaveTags" and "LoadTags" control templates do not suit your needs, you can utilize this template to create your own solution.

TagRecord - This Code template takes the current record in memory. It saves you the trouble of remembering how the syntax for calling the tagging library.

UntagRecord - This Code template is a companion of "TagRecord". It untags the current record in memory.

FlipRecordTag - This Code template is the third in the set including "TagRecord" and "UntagRecord". It toggles the tag on the current record.

WipeTags - This Code template is similar to the above three templates, except that it untags all records in the specified tag set.

Tagging API - This collection of procedures and functions permit you full access to the tagging subsystem.

Multi-APP Support - Your systems can be comprised of a single APP/EXE, or many APPs/DLLs/EXEs. The tagging library is contained only in your base APP.

Upon entry into a Browse procedure, you can specify whether existing tags should be cleared automatically, whether they would be left as-is, or whether the user should be asked each time the browse is called.

After each TagOne, UntagOne and FlipOne operation in the BrowseBox, you can optionally have the scroll bar moved to the next record.

You also have control of the character (or string) used to "checkmark" the record in the BrowseBox. Or you can have icons represent the tagged/untagged status of the record. Your user can also click in the tag column to flip (toggle) the tag.

We've included optional icons for all of the tagging buttons, along with a one-click method for implementing them. We also optionally support the tagging operations on the popup. (If you are using the popup, you can even hide the buttons entirely.)

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.

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

<u>Installation Directory Structure</u>

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.

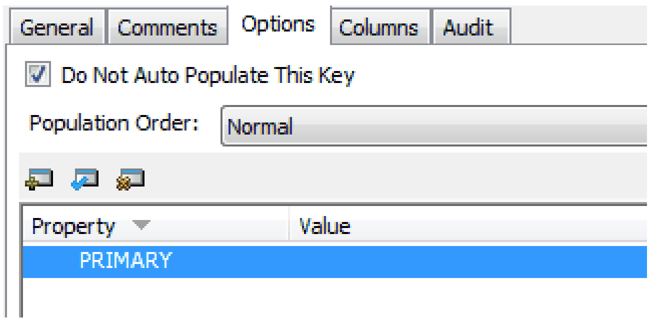
Assuming this all went without a hitch, you're ready to start using the templates.

1.4 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:



☑ Do Not Auto Populate This Key

Population Order: Normal

Property	Value
PRIMARY	

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

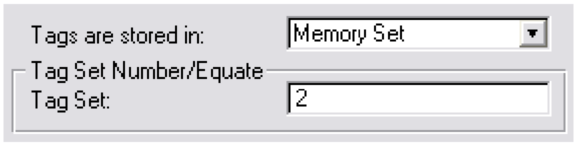
- Disk Set
- BYTE Field in Primary

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



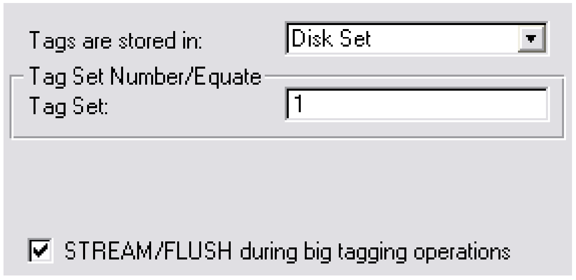
Tags are stored in: Memory Set
Tag Set Number/Equate
Tag Set: 2

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



Tags are stored in: Disk Set
Tag Set Number/Equate
Tag Set: 1
[✓] STREAM/FLUSH during big tagging operations

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.

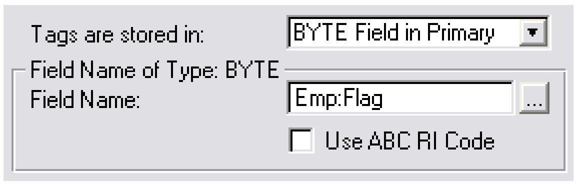
Tags are stored in either TagFile_ or TagFilePos_, depending on whether your primary key field is a numeric or string field. TagFile_ and TagFilePos_ are separate files usually using the TopSpeed file driver (or other driver of your choice).

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



Tags are stored in: BYTE Field in Primary
Field Name of Type: BYTE
Field Name: Emp:Flag
Use ABC RI Code

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.

1.5 Upgrading from Earlier Versions

These sections are in reverse chronological order.

Upgrading to Super Tagging 7.01

This version added support for TagRest, UntagRest and FlipRest buttons. When you populate a new instance of the template for your browse, you'll automatically get these buttons. For you existing browses, you can add them by copying the ?TagAll, ?UntagAll and ?FlipAll buttons, and then tweaking the windows source via the [...] button beside the [Window] button in the IDE.

When you copy a button, it will have many of the same settings as the original. However, you must make the following tweaks:

1. Edit the #ORIG attributes, specifying ?TagRest, ?UntagRest and ?FlipRest. The template uses this attribute to determines the specific purpose of each button (via %ControlOriginal). The value is case sensitive!
2. Add the #SEQ attribute matching the value on the other tagging buttons. The template uses this to determine which controls belong to it (via %ControlInstance).

When you're done, you'll have something like this in C6:

```
BUTTON('Tag &All'),AT(272,20,68,16),USE(?TagAll),#SEQ(5),#ORIG(?TagAll)
BUTTON('Unta&g All'),AT(272,40,68,16),USE(?UntagAll),#SEQ(5),#ORIG(?UntagAll)
BUTTON('F&lip All'),AT(272,60,68,16),USE(?FlipAll),#SEQ(5),#ORIG(?FlipAll)
BUTTON('Tag &Rest'),AT(344,20,72,16),USE(?TagRest),#SEQ(5),#ORIG(?TagRest)
BUTTON('Untag Rest'),AT(344,40,72,16),USE(?UntagRest),#SEQ(5),#ORIG(?UntagRest)
BUTTON('Flip Rest'),AT(344,60,72,16),USE(?FlipRest),#SEQ(5),#ORIG(?FlipRest)
```

In C7 it will look like this:

```
BUTTON('Tag &All'),AT(272,20,68,16),USE(?TagAll),#ORIG(?TagAll),#SEQ(5),#ORDINAL(9)
BUTTON('Unta&g All'),AT(272,40,68,16),USE(?UntagAll),#ORIG(?UntagAll),#SEQ(5),#ORDINAL(10)
BUTTON('F&lip All'),AT(272,60,68,16),USE(?FlipAll),#ORIG(?FlipAll),#SEQ(5),#ORDINAL(11)
BUTTON('Tag &Rest'),AT(344,20,72,16),USE(?TagRest),#ORIG(?TagRest),#SEQ(5),#ORDINAL(12)
BUTTON('Untag Rest'),AT(344,40,72,16),USE(?UntagRest),#ORIG(?UntagRest),#SEQ(5),#ORDINAL(13)
BUTTON('Flip Rest'),AT(344,60,72,16),USE(?FlipRest),#ORIG(?FlipRest),#SEQ(5),#ORDINAL(14)
```

Upgrading to Super Tagging 6.0

The new thread handling in Clarion 6 required that we change some of our methods.

The UsrTag_ procedure formerly required a *LONG parameter. The tagging system would save a reference to this, so that it automatically knew when the user number had changed. Now it takes a LONG value, and it merely remembers that value. If your original value changes, you must call UsrTag_ again to update it to the new value. (If you're also using Super Security, then this is automatically handled for you.)

Upgrading to Super Tagging 4.0

We have made a number of changes to the Super Tagging templates during our migration to Clarion 4+ABC. Please take note of the following:

- If you had not already imported the tagging files into your dictionary, you must do so now. The import file is SUPER\LIBSRC\TAGGING\TAGGING.TXD.
- We are no longer using Glo::Pointer and Glo::Position. You can delete these variables from your Global Data.
- We have managed to remove all settings from the global extension. (The extension itself is still required.) The "Flag Field Tagged Value" setting has been moved to the individual procedure templates. The Library location is determined by your Global Properties setting for "Generate template globals and ABC's as EXTERNAL". If this is OFF then the tagging library is internal, and vice versa.

Tag Storage Changes

We first added tagging features to the SuperModels for Clarion 2.0 for DOS. At that time only Clarion datafiles were supported and the concept of a "primary key" wasn't very common in the Clarion world. Therefore, we decided to use POINTER(File) instead. With the addition database drivers in Clarion 3.0 for DOS, we added support for POSITION(File). In our templates for Clarion for Windows, we added BYTE Field in Primary. After this, we also added support for using both numeric and string primary key fields, and also for POSITION(PrimaryKey). Finally, we added support for storing these tags in Memory.

With the release of Super Tagging 4.0, we have reduced these storage methods from eleven to five, and we've simplified their usage!

Clarion 4 (ABC) has enforced the use of VIEWs as the preferred method of file access. For us to support POINTER(File), POSITION(File) and POSITION(PrimaryKey), we would have had to make excessive use of the REGET operation. At the very least, this would have made tagging operations very slow. Instead, we decided that support for these methods is not longer necessary.

Because of this decision, you will be required to have a primary key in your file with a single field component. (The inclusion of a primary key of this type is good database practice, anyway.) This primary key field can be an integer (e.g.: LONG) or a string. If the field is an integer, then tags will be stored in TagFile_. Otherwise, they will be stored in TagFilePos_.

When you run your APP through Clarion's Conversion Wizard, it will automatically make the modifications to your tag storage settings. "TagFile_ (POINTER)" and "TagFilePos_ (POSITION)" are changed to "Disk Set"; "Memory (POINTER)" and "Memory (POSITION)" are changed to "Memory Set"; while "BYTE Field in Primary" is left as-is.

NOTE: If you don't care about existing saved tags, you can just delete TAGFILE_.TPS, TAGFILEP.TPS and TAGSET_.TPS. If you have always set "Reference Source" to "Primary Key", then your tags are already compatible with the new system. If neither of these applies, then you will have to create a tag conversion program.

The conversion program is created by a Utility template included with Super Tagging 2.5. Unfortunately, the utility templates cannot access all of your various settings, so you will have to help it along.

Start by loading your application with the legacy templates. Press Ctrl-U (or select "Application /

Template Utility" from the pulldown). You'll find two template utilities under "Class Super Tagging":

TagTemplateFinder - This Utility template is used to list all procedures which incorporate tagging operations, so that you can record those settings and enter them into the TagStorageConvertor.

TagStorageConvertor - This Utility template creates a PRJ+CLW to convert your tags from unsupported legacy storage methods to the new methods. It automatically scans your APP for a list of the datafiles involved in tagging operations. You must specify the tag storage method for each of these files, plus any tag set numbers that need to be converted. Saved tags are handled for all specified files.

NOTE: This conversion program assumes that you have been consistent with your tag storage settings:

- You must never use the same tag set number for two different files.
- For a particular file, you must specify a consistent setting for "Tags are stored in", "Reference Source", and "Use POSITION(PrimaryKey)". Otherwise, the convertor will jumble your tags.

The template will use your global extension setting for "Tag File Location" to determine whether it should generate the tag support files itself, or produce them from the dictionary.

Once you've created the conversion program, you need only load the project and compile it into an EXE. Of course, you should tag some records in the legacy version of your program, run it through the conversion wizard, then check it against the converted tags.

2 Template Usage

2.1 Adding Tagging to your Applications

The Super Tagging templates are actually a combination of templates and functions. The templates should handle all interfacing with the functions, so you should never have to call them yourself. If you are adventurous, though, you can certainly give it a try. (For more information on this, see API Reference.)

If you are using the full Super Tagging, you must begin by importing one TXD into your dictionary. Then you add the Global Support Extension template, followed by any number of other templates throughout your application.

For more information, see:

Support Files
Global Extension Template
Tag Records in a BrowseBox
Browse Marking (Windows Style)
Optionally Filter Tagged Record in a BrowseBox
Filter Tagged Records in a BrowseBox
Filter Tagged Records in a Process or Report
Get Tagged Record in Process/Report
Save Tags (Control Template)
LoadTags (Control Template)
Process Tagged Records (Code Template)
Copy Tags from One Tag Set to Another (Code Template)
Tag/Untag/Flip Current Record (Code Templates)
Wipe All Tags (Code Template)
Open and Close TagFile

2.2 Support Files

(TAGGING.TXD)

In Super Tagging 4.0, we decided to always place the tagging files in the dictionary. It gives us many benefits, and clears up much confusion regarding the settings for the global extension template.

Importing the Tagging Files

The first step is to load your dictionary, and import TAGGING.TXD from SUPER\LIBSRC\TAGGING. You'll end up with the following files in your dictionary:

TagFile_ This file stores tags when you are using "Disk Set" with a file whose primary key field is an integer (i.e.: LONG). In addition to the reference to the data record, it also contains the tag set number and user (when applicable).

TagFilePos_ This file stores tags when you are using "Disk Set" with a file whose primary key field is not an integer (i.e.: STRING). In addition to the reference to the data record, it also contains the tag set number and user (when applicable).

TagSet_ This file contains the names for the saved tag sets. Each record contains the tag set (always a negative number), source file name, count, and user (when applicable).

2.3 Global Extension Template

To use the Tagging features, you must add the global support to your application. The TagGlobal extension template can be found within the "Class SuperTagging".

There are no global settings for this template! For those of you familiar with old Super Tagging versions, the following settings apply:

Support tagging using TagFile_ (POINTER) - ON

Support tagging using TagFilePos_ (POSITION) - ON

Omit ST::SelectTagSet Procedure - OFF

Location of Library - This is determined by your setting of "Generate template globals and ABC's as EXTERNAL". If this setting is ON, then the tagging support library is external. Otherwise, it's internal.

Location of File Definitions - As of Super Tagging 4.0, you must import all three tagging files (TagFile_, TagFilePos_, and TagSet_) into your dictionary.

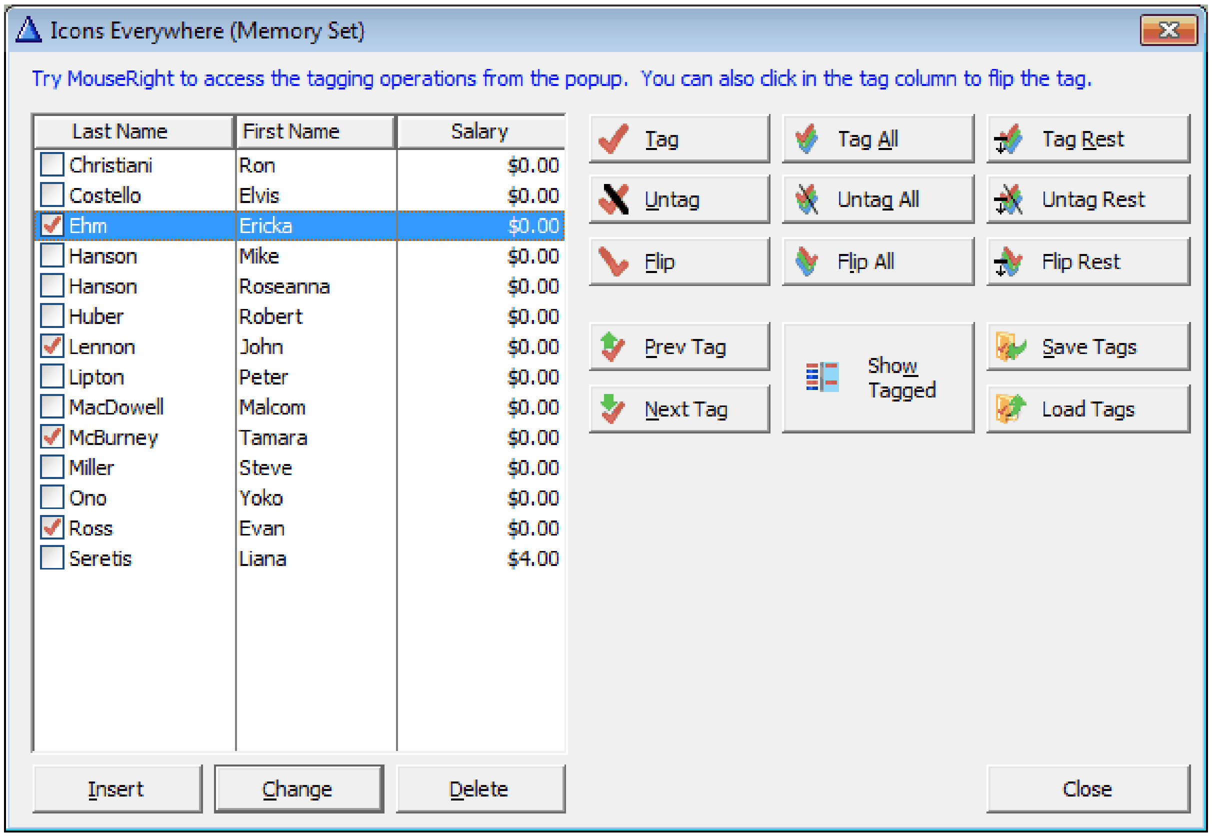
POSITION() Size - 1024

Flag Field Tagged Value - This setting has moved to the individual Control, Extension, and Code templates.

2.4 Tag Records in a BrowseBox

(Control Template)

New in 7.01 - We added buttons to Tag, Untag and Flip the *Rest* of the records. These are essentially the same as the *All* buttons, except that they starts at the currently highlighted record. They're handy if you have a large number of records, and you want to process them in separate batches. For example, you start by tagging all, jump to a place lower on the list and untag rest, then run your process on the first batch. Then you untag all, go to that same location, tag the rest, and then run your process on the remaining records. To learn how to add these buttons to your existing browses, see Upgrading from Earlier Versions.



Try MouseRight to access the tagging operations from the popup. You can also click in the tag column to flip the tag.

	Last Name	First Name	Salary
☐	Christiani	Ron	$0.00
☐	Costello	Elvis	$0.00
✓	Ehm	Ericka	$0.00
☐	Hanson	Mike	$0.00
☐	Hanson	Roseanna	$0.00
☐	Huber	Robert	$0.00
✓	Lennon	John	$0.00
☐	Lipton	Peter	$0.00
☐	MacDowell	Malcom	$0.00
✓	McBurney	Tamara	$0.00
☐	Miller	Steve	$0.00
☐	Ono	Yoko	$0.00
✓	Ross	Evan	$0.00
☐	Seretis	Liana	$4.00

Adding tagging support to a browse is very simple. Your window must already contain a Clarion BrowseBox Control. You must perform the following steps:

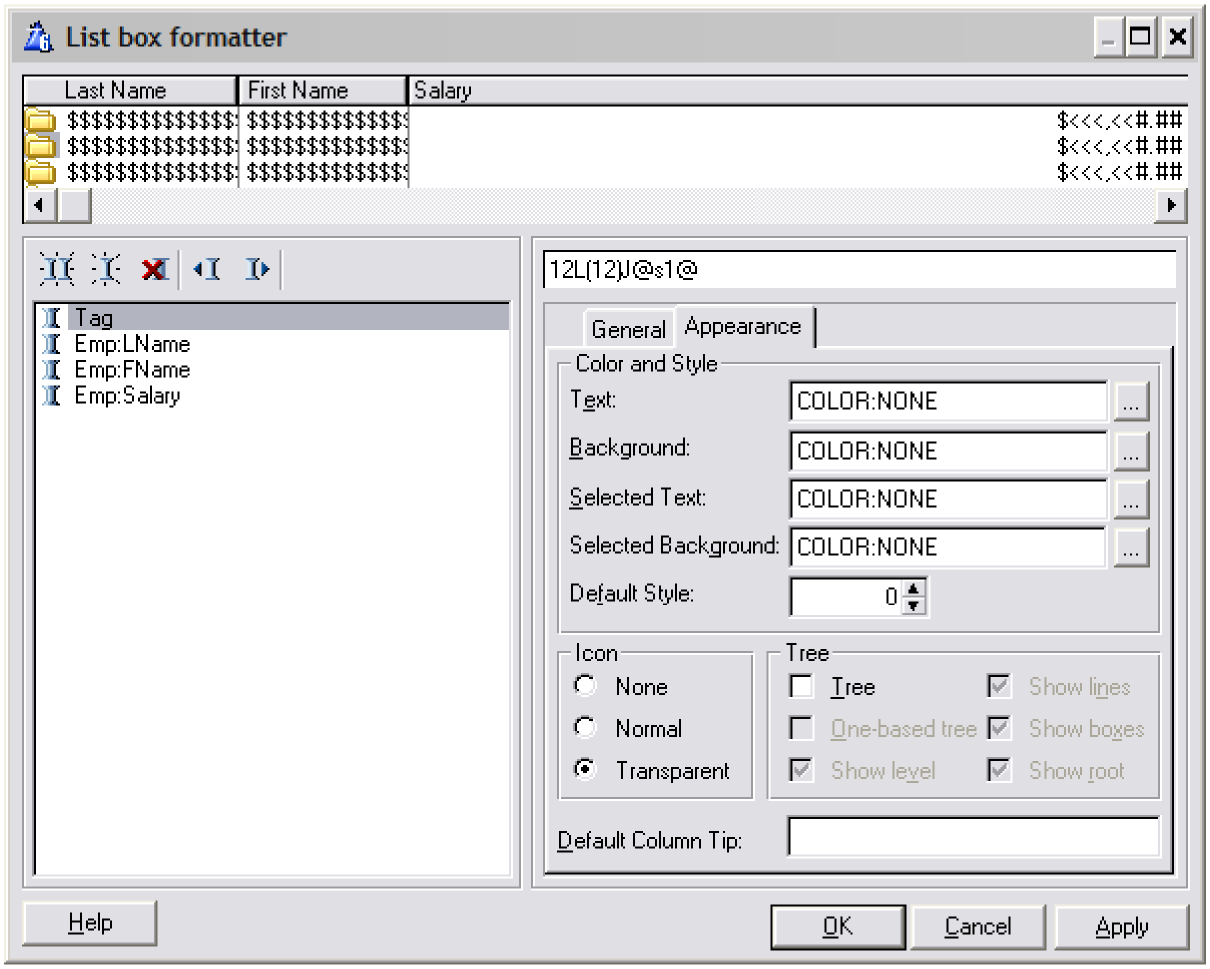
1. Populate the Control Template
2. Modify the BrowseBox Format to add the Tag column
3. Enter the BrowseTagging Extension's Prompts
4. Add Icon support for the Tag Column

Populate the Control Template

1. Highlight your existing Browse procedure in the "Application Tree", then press the [Properties] button.
2. Press the [Window] button to edit the window structure.
3. Expand the bottom of your window to make room for the controls. (If there is not enough space, the screen editor will automatically expand it for you when you populate the control.)
4. Press the "Control Template" button on the "Controls" Toolbox, or select the "Populate / Control Template..." option from the pulldown menu.
5. Locate "Class SuperTagging", then "BrowseTaggingButtons" below that. Highlight this option and press the [Select] button.
6. As you move your cursor across your window structure, you will notice the cursor changes to the "Populate Crosshairs". Place the cursor below the BrowseBox and near the left side of the window, then click the left mouse button.
7. The template will automatically populate eight buttons onto the window: "Tag", "Untag", "Flip", "Tag All", "Untag All", "Flip All", "Prev Tag", and "Next Tag".
8. If you wish, you can delete any of these buttons. When you do, the editor will ask, "Do you want to remove the control template?". You would normally answer "No", because this will remove the tagging support. (If you removed all of the buttons, the control template would be removed from your procedure.)
9. If you wish to group the buttons within a box, then you must perform a minor trick: To tell Clarion to group fields, you must move them into a group box. It is not enough to drag a group box over existing fields. Here's how you do it:
    a. Expand the bottom of the window to make space.
    b. Press the "Group Box" button on the "Controls" Toolbox, or select the "Control / Group Box" option from the pulldown menu.
    c. Position the "Populate Crosshairs" below the tagging buttons, then press the left mouse button to create the group box.
    d. Expand the group box so that it will hold all of the buttons.
    e. Select the buttons using Ctrl+Drag (or Shift-Click), then drag and drop them within the group box.
    f. Adjust positioning and size as desired.

Once you get a feeling for the size of the default tagging buttons, you can pre-populate the group box, then populate the tagging buttons within it.

Modify the BrowseBox Format



12L(12)J@s1@

Color and Style

Text:	COLOR:NONE
Background:	COLOR:NONE
Selected Text:	COLOR:NONE
Selected Background:	COLOR:NONE
Default Style:	0

Icon: ( ) None ( ) Normal [✓ Transparent]

Tree: [ ] Tree [✓ Show lines] [ ] One-based tree [✓ Show boxes] [✓ Show level] [✓ Show root]

Default Column Tip:

1. Now you need to add the tag marker to your browse box. Click on the BrowseBox, right-click your mouse, then select the "List Box Format..." option from the pop-up window.
2. Highlight the first column in the "List Box Formatted" window, then press the [Populate...] button.
3. Highlight the "LOCAL DATA" entry in the left box, highlight the "Tag" field in the right box, the press the [Select] button.
4. The default format is Left-Justified with @S10, and Heading Text of "Tag". We suggest that you change the width to 11, the picture to @S1, and the header justification to Center, and data justification to Left. We also remove the heading text. If you are using a multi-character tag mark (see next section), then you will want to make this field wider. If you wish to use an icon in the tag column, there is a setting to achieve this automatically.
5. Press the [OK] button to save the "List Field Properties".
6. Now you will probably want to move the column. Press the "shift left" button to shift it to

column one.

7. Press the [OK] button to exit the "List Box Formatter" window.

8. Finally, press [OK] to save the "List Properties".

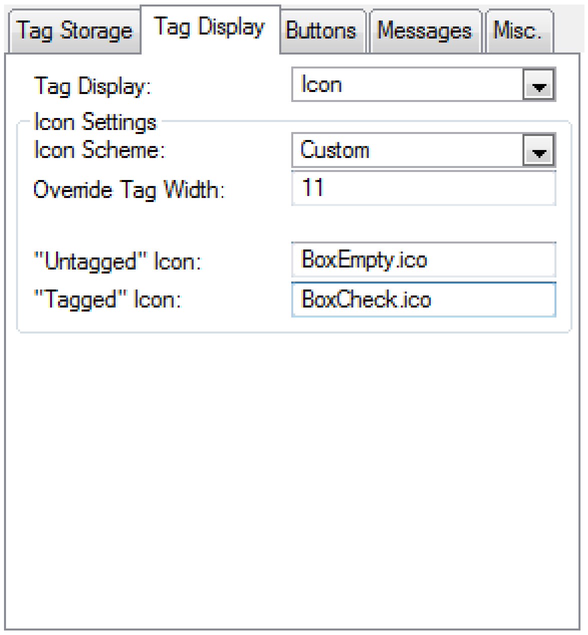
Enter the BrowseTagging Extension's Prompts

To access these prompts, press [Extensions] button on the "Procedure Properties" window. Then highlight the "Tag Records from Browse Box on..." entry. If you have multiple BrowseBoxes with tagging on this window, you'll have to find the proper one for your desired primary file.

Tag Storage Tab

See the "Tag Storage Options" in the Introduction.

Tag Display Tab



Tag Display: Icon
Icon Scheme: Custom
Override Tag Width: 11
"Untagged" Icon: BoxEmpty.ico
"Tagged" Icon: BoxCheck.ico

Tag Display - Choose whether you want to see Text or an Icon to represent the tagged status. If you want to use an icon, and you had already manually configured your icons using a prior version of Super Tagging, then you must eliminate those icon options in the Browse settings. This involves deleting all conditional icons, and blanking the standard icon for the Tag column.

Tag Character - If you chose "Text" above, this is the character that is displayed in the BrowseBox when a record is tagged. The default is an asterisk "*". You may want to change this to a large "X".

You can also use a multi-character string, up to ten characters long. Unless your using

Styles, the tag column shares the same font as the rest of the browse box, so you can't use a checkmark without either using a custom style or changing the entire BrowseBox to another font that supports a checkmark character.

Your can also use a variable name. To do this, you must prefix the variable name with an exclamation point (!).

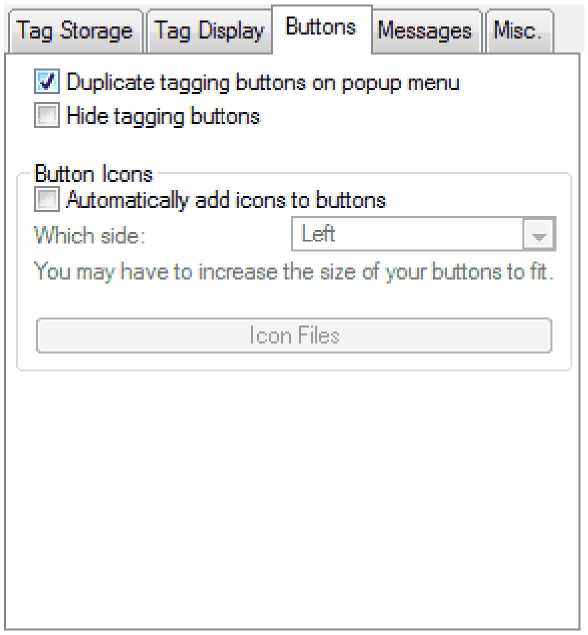
Icon Scheme - If you chose "Icon" above, this is the icon combination that you wish to utilitize. If you are not happy with any of the standard options, you may choose "Custom", and specify your own icons below. Other than the standard schemes, there are many other sample icons provided in the SUPER\IMAGES directory. Of course, you can also use icons of your own design.

Override Tag Width - This forces the width of the Tag column at run-time. If you leave it blank, it defaults to ?ListControl{PROPLIST:LineHeight}. If desired, you can enter a constant value or expression.

"Untagged" Icon - This is the icon used to represent the "Untagged" status of the record.

"Tagged" Icon - This is the icon used to represent the "Tagged" status of the record.

Buttons Tab



[✓] Duplicate tagging buttons on popup menu
[ ] Hide tagging buttons

Button Icons
[ ] Automatically add icons to buttons
Which side: Left
You may have to increase the size of your buttons to fit.

Duplicate tagging buttons on popup menu - If you turn this on, then the various tagging buttons will be represented on the popup that appears when you right-click on the list box. (The Popup support must turned of in the BrowseBox extension settings.) Any tagging buttons that have been deleted will not appear on the popup.

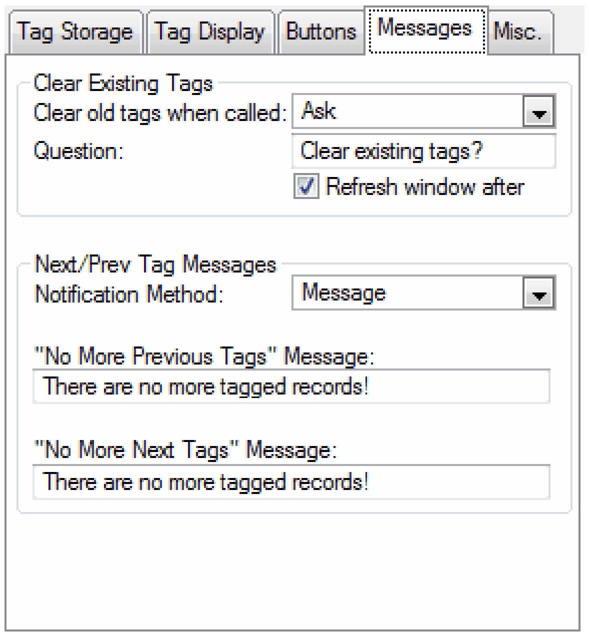
Hide tagging buttons - If the POPUP option is on, then this option will hide the tagging buttons at run-time. Just move the buttons into the list box area to free up the space.

Automatically add icons to buttons - If you didn't hide the buttons above, then you can have the system automatically add icons to each of the tagging buttons that are on the window.

Which side - This specifies whether the icons should appear to the left, right or default (top) of the button text. If you set the button width to "Default" and the side to "Left", then the text will automatically be hidden.

Icon Files - You can override the default icons to be used for the tagging buttons. These files are stored in the SUPER\IMAGES directory. There are only one set of icons for TagOne, UntagOne, FlipOne, TagRest, UntagRest, FlipRest, NextTag and PrevTag. There are two sets for TagAll, UntagAll and FlipAll; these files are TAGALL2,ICO, UNTAGAL2.ICO and FLIPALL2.ICO. If you are not happy with these icons, feel free to specify your own. Enter the filename without a tilde (~).

Messages Tab



Clear Existing Tags
Clear old tags when called: Ask
Question: Clear existing tags?
✓ Refresh window after

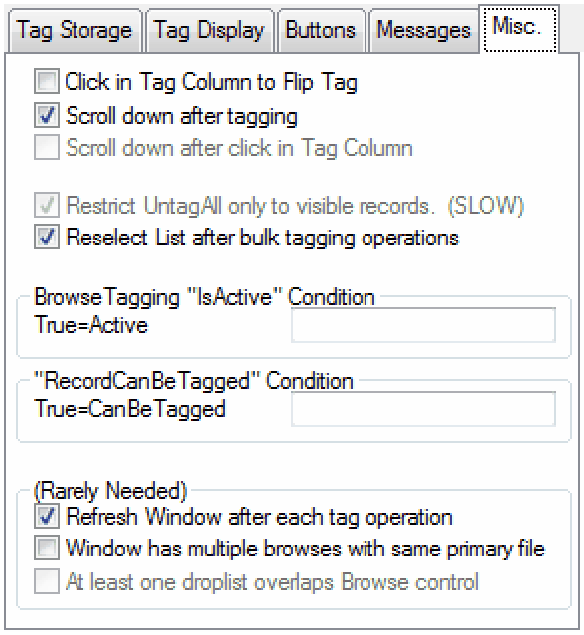
Next/Prev Tag Messages
Notification Method: Message
"No More Previous Tags" Message: There are no more tagged records!
"No More Next Tags" Message: There are no more tagged records!

Clear Existing Tags - These settings control what happens when you call a BrowseBox when there are tags remaining from a prior tagging session. Your options are:

Yes	Always clear existing tags automatically.
No	Never clear existing tags.
Ask	Ask the user what they would prefer to do. The Question field is used to control the appearance of the screen when it pops-up. The question can be a string (without quotes) or a variable name (prefixed by an exclamation point).

Prev/Next Tag Messages - When a user presses the [Prev Tag] and [Next Tag] buttons, and there are no more tags in that direction, then the user will be notified. They can be shown a message, hear a beep or get both the beep and the message. The "Message" entry can be a string (without quotes) or a variable name (prefixed by an exclamation point). The "Message" is required.

Misc. Tab



- [ ] Click in Tag Column to Flip Tag
- [x] Scroll down after tagging
- [ ] Scroll down after click in Tag Column
- [x] Restrict UntagAll only to visible records. (SLOW)
- [x] Reselect List after bulk tagging operations

BrowseTagging "IsActive" Condition
True=Active

"RecordCanBeTagged" Condition
True=CanBeTagged

(Rarely Needed)
- [x] Refresh Window after each tag operation
- [ ] Window has multiple browses with same primary file
- [ ] At least one droplist overlaps Browse control

Click in Tag Column to Flip Tag - This allows your users to click in the tag column to toggle the tagged/untagged status.

Scroll down after tagging - When you are performing individual tagging operations (like Tag One, Untag One, and Flip One), you may want the selector bar to move to the next entry in the List.

Scroll down after click in Tag Column - If you want to have the selector bar to move to the next entry when clicking in the tag column, check this box.

Restrict UntagAll only to visible records - By default, the UntagAll operation will untag only those records that are accessible by the current Range and Filter criteria. This can be slow, as it has to walk through each individual record, checking if it is tagged. If you turn this off, the NewTag procedure is used, which very quickly deletes all tagged records for the entire file.

Reselect List after bulk tagging operations - This causes the input focus to return to the list box after you've hit one of the bulk tagging buttons. If this is off, focus will remain on the pressed button.

BrowseTagging "IsActive" Condition - This optional expression affects all tagging operations on the window. If it's TRUE, then tagging is supported as usual. If it's FALSE then all tagging operations are turned off.

"RecordCanBeTagged" Condition - This optional expression selectively disables tagging on a per-record basis. The tagging buttons and pulldown will still be enabled, but the actual tagging operation will not occur.

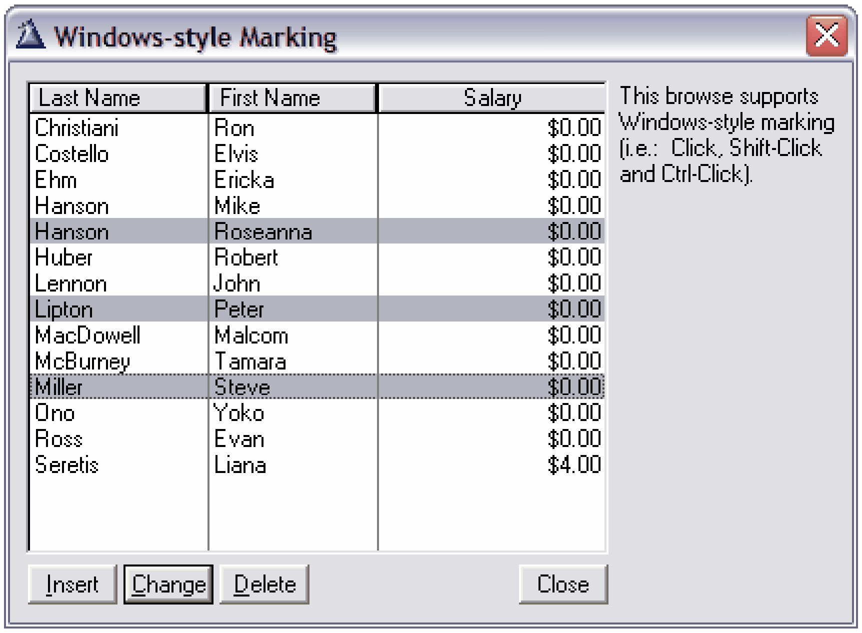
Refresh Window after each tag operation - By default, the window is not refreshed after each tagging operation. This is done for speed reasons. If, however, you have fields that can change with each bit of tagging activity, then you should turn this on. A good example of this is a computed field that uses CntTag to display the current number of tagged records.

Window has multiple browses with same primary file - If you've got more than one browse that uses the same file as the tagging browse, then the tagging system needs to take special precautions to ensure that the proper record is loaded, corresponding with the highlighted record in the tag browse. However, this takes longer than usual, so you can optionally invoke this here.

At least one droplist overlaps Browse control - If you have one or more drop list controls that could overlap the tagging column on the browse control, then clicking those items could confuse the code that handles "Click in Tag Column to Flip Tag". In that situation, turn this ON to tell the tagging system to use an alternative method to differentiate between the two operations.

2.5 Windows-Style Marking in a BrowseBox

(Extension Template)



Last Name	First Name	Salary
Christiani	Ron	$0.00
Costello	Elvis	$0.00
Ehm	Ericka	$0.00
Hanson	Mike	$0.00
Hanson	Roseanna	$0.00
Huber	Robert	$0.00
Lennon	John	$0.00
Lipton	Peter	$0.00
MacDowell	Malcom	$0.00
McBurney	Tamara	$0.00
Miller	Steve	$0.00
Ono	Yoko	$0.00
Ross	Evan	$0.00
Seretis	Liana	$4.00

This browse supports Windows-style marking (i.e.: Click, Shift-Click and Ctrl-Click).

This template provides Windows Standard Behavior (WSB) marking. This means that your users can click, shift-click and ctrl-click to mark records in a regular Clarion BrowseBox. This should not be used with a large data set (i.e.: more than a few hundred records available in the current browse), as the Shift-Click processing uses a brute force method that can be slow for large numbers of records. If your browse has a filter and/or range that restricts the available records, then this will not be a problem.

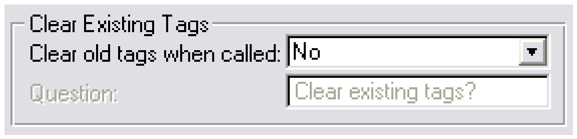
To add the extension, perform the following steps:

1. Go to the Extensions window of your Procedure Properties.
2. Highlight the BrowseBox to which you wish to add marking.
3. Press [Insert] to add a new extension template.
4. Locate "Class SuperTagging", then "BrowseMarking" below that. Highlight this option and press the [Select] button.
5. Enter the extension prompts.

<u>Tag Storage Tab</u>

See the "Tag Storage Options" in the Introduction.

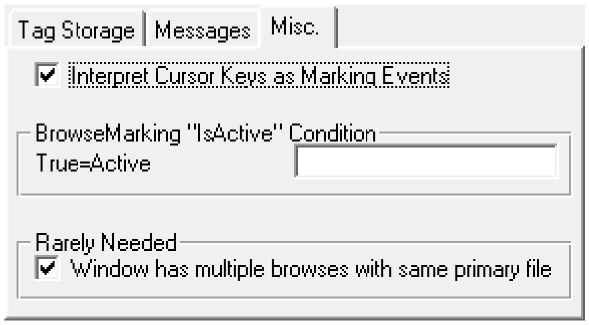
Messages Tab


Clear Existing Tags
Clear old tags when called: No
Question: Clear existing tags?

Clear Existing Tags - These settings control what happens when you call a BrowseBox when there are tags remaining from a prior tagging session. Your options are:

Yes	Always clear existing tags automatically.
No	Never clear existing tags.
Ask	Ask the user what they would prefer to do. The Question field is used to control the appearance of the screen when it pops-up. The question can be a string (without quotes) or a variable name (prefixed by an exclamation point).

Misc. Tab



☑ Interpret Cursor Keys as Marking Events
BrowseMarking "IsActive" Condition
True=Active
Rarely Needed
☑ Window has multiple browses with same primary file

Interpret Cursor Keys as Marking Events - If the user presses the Up/Down/PgUp/PgDn/Home/End keys, this will automatically treat the newly selected record as if it was Clicked. In other words, all marks will be canceled, and the newly selected record will be the only marked record.

If, however, the user holds down the Shift key while moving about with the cursor keys, then the operations are treated as a Shift-Click. It will automatically mark all records from the most recent "anchor" (i.e. clicked record) up/down to the newly highlighted record.

NEW in Super Tagging 6.63: Ctrl+Space toggles the marked status for the currently highlighted record. Ctrl+Backspace unmarks all. The user may navigate without affecting the marked state by holding down the Ctrl key while pressing the cursor keys.

If you have this OFF, then the highlighter bar will move as a "phantom", without affecting the marked state of the records.

BrowseMarking "IsActive" Condition - If you want to support marking only when a certain condition is in effect, then specify that condition here. If it evaluates to TRUE, then marking will be active.

Window has multiple browses with same primary file - If you've got more than one browse that uses the same file as the marking browse, then the tagging system needs to take special precautions to ensure that the proper record is loaded, corresponding with the highlighted record in the marking browse. However, this takes longer than usual, so you can optionally invoke this here.

2.6 Optionally Filter Tagged Records in a BrowseBox

(Control Template)

Your users may wish to switch between viewing "all records", "only tagged records" and "only untagged record". The "BrowseOptTagFilter" Control template is designed with this goal in mind. It is populated onto the browse window as a button that changes its text and icon depending on the current viewing state. The extension settings are as follows:

Tag Storage Tab

See the "Tag Storage Options" in the Introduction.

Button Text Tab

Support "Show Untagged" Option - This enables the "Show Untagged" state.

Text Button - These are the phrases that will be displayed on the button. You can choose to place the text on the button, or in the tool tip (in case the button is large enough for only an icon).

The "Show Tagged" message is displayed in the button when they are viewing all records. By pressing the button, it will change to the "Show Untagged" or "Show All" text (depending on other settings), and they will be in another filter state. In each of these text settings, you can include an ampersand (&) to control the "hot key" for the button.

The final option is a drop box controlling the initial state when the user enters the browse. The default for this is "Show All".

Icons Button - If desired, you can have icons automatically displayed in the button. You can control whether they are displayed on the left, right or default (center). You also specify the icon files to be used.

Miscellaneous Tab

Switch to "Show All" with [Untag All] Button *(NEW in Super Tagging 6.03)* - In most cases, this template will be used alongside the tagging buttons control template. In those situations, it's helpful to have the filter mode reset to "Show All" if the [Untag All] button is pressed. Otherwise, the browse window will be empty, after the user waits for all records to be read and discarded.

In case you have more than one set of BrowseTaggingButtons, you must also specify the name of the "Untag All" control.

2.7 Filter Tagged Records in a BrowseBox

(Extension Template)

This Extension template is similar to the "BrowseOptTagFilter" control template. It is used to display only tagged records on a Browse. The only difference is that this template has a fixed filter, rather than giving the user the option of toggle between filtered and non-filtered. This extension can be used only with a procedure already containing a BrowseBox.

Tag Storage Tab

See the "Tag Storage Options" in the Introduction.

If your browse used "BYTE Field in Primary" as its tag storage method, then you don't need to use this extension. Instead, you could use a key on the tag field to make the Browse very fast. This is not really necessary unless you have many records in your file. We'll leave that up to your personal preference.

2.8 Get Tagged Data Records in a Process or Report

(Extension Template)

This extension template is used when you are running a Report or Process directly against TagFile_ or TagFilePos_. It automatically retrieves the tagged record based upon the saved Ptr/ Pos value. The reports and processes are very fast, because only the tagged records are retrieved. The template automatically sets the filter for the view. If your data file is related to your tag file, you can also specify the processing order.

To use the Order feature, your data file must be related to your TagFile. Set up a relationship between either TagFile_ (if your primary key field is an integer) or TagFilePos_ (if your primary key field is a string). The data file will be on the "ONE" side, with the TagFile on the "MANY" side. Do not specify a foreign key for the TagFile. The data file will use its primary key. Equate the data file's primary key field with the Ptr/Pos field from TagFile.

The primary file for the Process or Report will be TagFile_ or TagFilePos_. Do not specify a key. Add the data file as a secondary file to the TagFile. This file must be the same as the original file with the tagged records.

Next you need to add the ProcessGetTaggedData extension template. Once you've done this, you will see the following settings:

Tag Storage Tab

See the "Tag Storage Options" in the Introduction.

You must also enter the data file to access. This will be your datafile (which has been specified as secondary to the TagFile in the file schematic.)

Order Tab

You can specify the processing order here. Enter the clause manually, or have the template assist you. If you choose to do it manually, the order clause will takes the following form:

```
+Cus:LastName,+Cus:FirstName,-Cus:Birthday
```

As an alternative, you can specify a field name so that the order is determined by your program logic at run-time. To do this, prefix the field name with an exclamation point (!):

```
!Loc:Order
```

If you would rather use assisted entry, then you can specify each field component and its order by picking it from the fields list. Make sure you include only fields that are part of the TagFile file schematic.

Miscellaneous Tab

Wipe tags after completion *(New in Super Tagging 6.11)* - Turn this on to clear the tags after the process/report is complete.

Perform Secondary Fetches for SQL - The REGET command sometimes doesn't seem to handle secondary files properly, especially with SQL files. For SQL, or if the process doesn't seem to be updating the records properly, try turning this switch ON.

2.9 Filter Tagged Records in a Process or Report

(Extension Template)

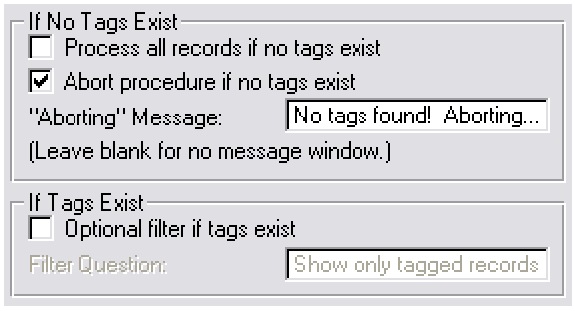
Once your users have tagged records, they will want to do something with them. The most common request is to print only tagged records in a report. Or they may want to delete the tagged records using a Process procedure. You can do this by adding this extension to your procedure.

Tag Storage Tab

See the "Tag Storage Options" in the Introduction.

If your browse used "BYTE Field in Primary" as its tag storage method, then you don't need to use this extension. Instead, you could use a key on the tag field to make the Browse very fast. This is not really necessary unless you have many records in your file. We'll leave that up to your personal preference.

Optional Filter Settings Tab



If No Tags Exist
- [ ] Process all records if no tags exist
- [x] Abort procedure if no tags exist

"Aborting" Message: No tags found! Aborting...
(Leave blank for no message window.)

If Tags Exist
- [ ] Optional filter if tags exist

Filter Question: Show only tagged records

If No Tags Exist:

Process all records if no tags exist - If there are no tags when the report/process is called, then it automatically processes all records if this is turned ON. If you turn this OFF, then no records will be processed.

Abort procedure if no tags exist - If the previous setting is OFF, then this option controls whether the procedure should immediately abort, or to continue (even though no records will be processed).

"Aborting" Message - If the Abort setting is ON, then you can specify an optional message here.

If Tags Exist:

Optional filter if tags exist - In some cases you may want to give the user a choice of whether to process all records, or only tagged records.

Filter Question - If the previous setting is ON, then enter the question to be posed to the user.

Miscellaneous Tab

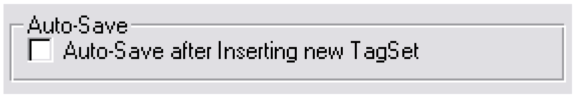
Wipe tags after completion *(New in Super Tagging 6.11)* - Turn this on to clear the tags after the process/report is complete. This setting respects the above settings, with regard to the optional processing of tags. For example, if you have the "Optional filter if tags exist", and the user answers "No", then the tags will not be wiped, even if this setting is true.

2.10 Save Tags for Future Retrieval (Control Template)

(Control Template)

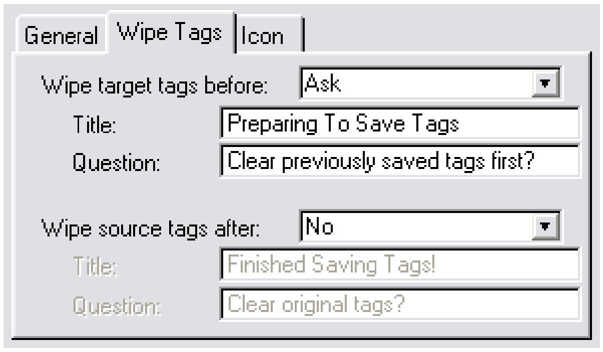
Your users may wish to save tags into sets for future retrieval. This is especially handy if you normally save your tags in memory and wish to occasionally save tags for future work. The "SaveTags" Control template is designed with this goal in mind. It is populated onto the browse window as a button and works in conjunction with the "LoadTags" control template. It is only available if you already have the BrowseTaggingButtons or BrowseMarking template populated on the window. The extension settings are as follows:

General Tab


Auto-Save
☐ Auto-Save after Inserting new TagSet

Auto-Save after Inserting new TagSet - When you users creates a new tag set name for saving, they probably want to immediately save the current set of tags with that name. If you prefer it to be a two-step operation (as it was before), then leave this OFF.

Wipe Tags Tab

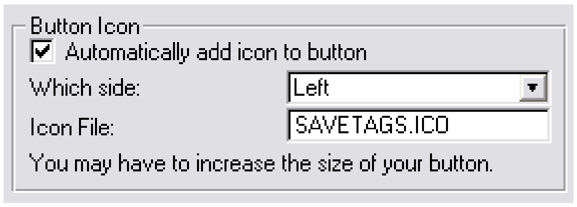


Wipe target tags before:	Ask
Title:	Preparing To Save Tags
Question:	Clear previously saved tags first?
Wipe source tags after:	No
Title:	Finished Saving Tags!
Question:	Clear original tags?

Wipe target tags before - Think of this as an "Overwrite saved file?" warning. If the user attempts to save into a set which already contains tags, they can optionally be warned of this. The options are "Ask" (the default), "Yes" (automatically wipe tags), and "No" (keep existing tags and add new ones).

Wipe source tags after - After the save is completed, you can clear the tags from the Browse itself. The options are "Ask", "Yes" (automatically wipe the tags), and "No" (the default).

Icon Tab



Button Icon

[✓] Automatically add icon to button

Which side: Left

Icon File: SAVETAGS.ICO

You may have to increase the size of your button.

Automatically add icon to button - You can have the system automatically add an icon to the SaveTags button.

Which side - This specifies whether the icon should appear to the left, right or default (top) of the button text. If you set the button width to "Default" and the side to "Left", then the text will automatically be hidden.

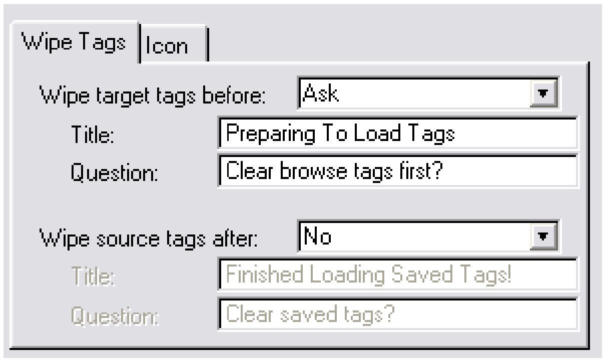
Icon File - The default is SAVETAGS.ICO. If you are not happy with this icon, feel free to specify your own. Enter the filename without a tilde (~).

2.11 Load Tags that were Saved (Control Template)

(Control Template)

Your users may wish to save tags into sets for future retrieval. This is especially handy if you normally save your tags in memory and wish to occasionally save tags for future work. The "LoadTags" Control template is designed with this goal in mind. It is populated onto the browse window as a button and works in conjunction with the "SaveTags" control template. It is only available if you already have the BrowseTaggingButtons or BrowseMarking template populated on the window. The extension settings are as follows:

Wipe Tags Tab



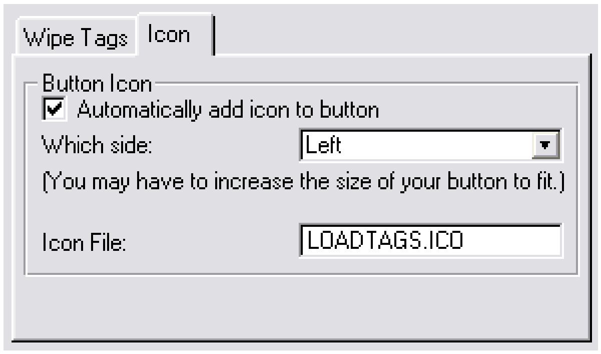
Wipe target tags before: Ask
Title: Preparing To Load Tags
Question: Clear browse tags first?

Wipe source tags after: No
Title: Finished Loading Saved Tags!
Question: Clear saved tags?

Wipe target tags before - Think of this as a "Discard Current Tags?" warning. If the user attempts to load when there are already tags present, they can optionally be warned of this. The options are "Ask" (the default), "Yes" (automatically wipe tags), and "No" (keep existing tags and add new ones).

Wipe source tags after - After the load is completed, you can clear the tags from the saved tag set. The options are "Ask", "Yes" (automatically wipe the tags), and "No" (the default).

Icon Tab



☑ Automatically add icon to button
Which side: Left
(You may have to increase the size of your button to fit.)

Icon File: LOADTAGS.ICO

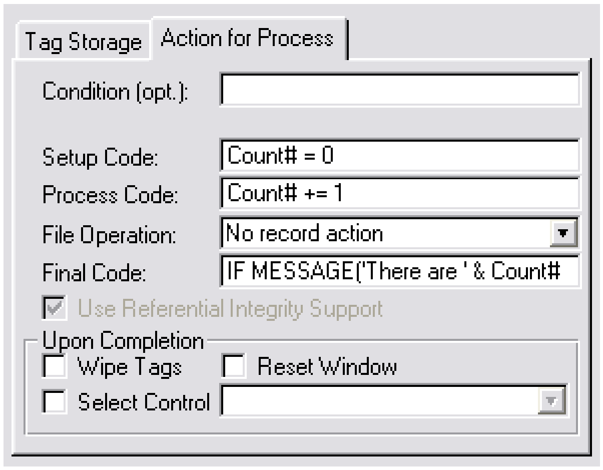
Automatically add icon to button - You can have the system automatically add an icon to the LoadTags button.

Which side - This specifies whether the icon should appear to the left, right or default (top) of the button text. If you set the button width to "Default" and the side to "Left", then the text will automatically be hidden.

Icon File - The default is LOADTAGS.ICO. If you are not happy with this icon, feel free to specify your own. Enter the filename without a tilde (~).

2.12 Process Tagged Records (Code Template)

(Code Template)



Condition (opt.):

Setup Code: Count# = 0

Process Code: Count# += 1

File Operation: No record action

Final Code: IF MESSAGE('There are ' & Count#

[✓] Use Referential Integrity Support

Upon Completion

[ ] Wipe Tags [ ] Reset Window

[ ] Select Control

In some situations, you will not want create a full-fledged Process procedure to handle your tagged records. For those quick-and-dirty situations, we provide a Code template called "Process Tagged". It simply walks through the tagged records and executes your specified code. It can optionally PUT or DELETE the record after this code.

You will be inserting the Code template into an Embed. In our example applications in C:\CLARION4\SUPER\EXAMPLES\TAGGINGx, we use it to display a count of tagged records when the user presses a button.

1. First we create a regular button with the text "Count Tagged Records".
2. Right-Click the mouse on the button, select "Actions..." option, then press the [Embeds] button.
3. Highlight the "Accepted" entry after "Control Event Handling, After Generated Code", then press the [Insert] button.
4. This brings you to the "Prompts for ProcessTagged" window. Here, again, we will specify the storage options for the tags.
5. New in Super Tagging 6.11 - If you wish to make this operation conditional, enter the Boolean expression in the "Condition" setting. This is wrapped around the entire operation, and NOT applied to each line.
6. Because we are counting something, we need to initialize a counter variable. Place the following code into the "Setup Code" prompt:

```
Count# = 0
```

7. Now you have to specify the "Process Code". If it is a simple process, you can just type the code here. Otherwise, you can "DO RoutineName", then create a routine in the "Procedure Routines" embed. In our case, we will simply issue the command:

```
Count# += 1
```

8. The next prompt is the "File Operation", which must be "No record action", "PUT record", or "DELETE record". In our case, we are not modifying the record.

9. Now that we've counted the tagged records, we need to display them. We'll do this in the "Final Code" field. Place the following code in the field:

   ```
   MESSAGE('There are ' & Count# & ' tagged records', 'Tag Count', ICON:Exclamation); DO SyncWindow
   ```

   The MESSAGE() procedure is used to display the count. The SyncWindow routine ensures that the table stays on the same record, and does not jump to the last tagged record.

10. The next option (ABC-only) is "Use Referential Integrity Support". If this is ON, then it will use Relate:Primary.Update() and Relate:Primary.Delete(), instead of PUT(Primary) and DELETE (Primary), respectively.

11. The final options appear inside a box entitled "Upon Completion".

    a) The first is "Wipe Tags". Turn this on if you want to clear the tags after completing the process. Now press [OK] to save these options. (If you are using "BYTE field in Primary" as your tag storage method, it actually clears the tag field during the process itself, and not after it's done, to prevent the need for second pass.)

    b) The second is "Reset Window". Turn this on to call ThisWindow.Reset(True) (for ABC) or ForceRefresh=True; DO RefreshWindow (for Legacy/Clarion).

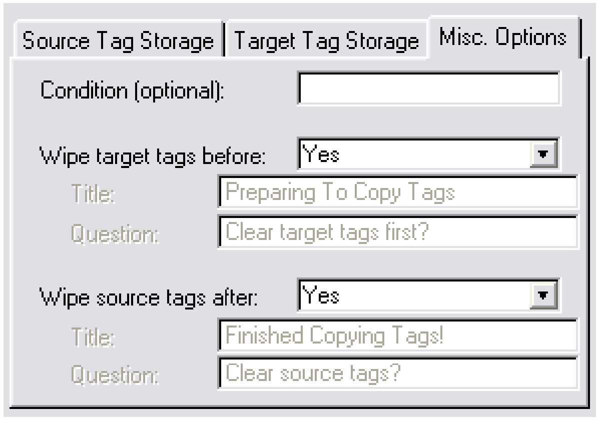
    c) The third is "Select Control". You can specify any control that you wish to receive focus (most likely a browse).

12. Return to the "Procedure Properties" window, and press the [Files] button. You'll see an entry that says "Process Tagged", with "<ToDo>" under it. Highlight the ToDo line, and press the [Insert] button. Now select the tagged file from the list.

13. Save the procedure and test your APP.

2.13 Copy Tags from One Tag Set to Another (Code Template)

(Code Template)



Condition (optional):

Wipe target tags before: Yes

Title: Preparing To Copy Tags

Question: Clear target tags first?

Wipe source tags after: Yes

Title: Finished Copying Tags!

Question: Clear source tags?

You can use the "CopyTags" Code template to copy tags from one tag set to another. There a number of situations where this would be handy. You could use the "BYTE Field in Primary" tag storage method for your day to day tagging operations, and you can save the tags using the "Disk Set" method. You would use one CopyTags template to copy from one set to another, then you would use another CopyTags template to get them back.

Another use is to "accumulate tags". Lets say that your users tag a few records each day in the process of working with the system. Once a day they copy the tags into a cumulative tag set. This tag set is processed once a month for customer statements, then cleared.

As you can see, there are many ways that you could use this feature. Here's how to set one up:

You will be inserting the Code template into an Embed. For example, you might connect it to a "Save Tags" button. This could copy the tags to a standard tag set, or it could ask the user for a tag set number to which they would be copied.

1. Choose an EMBED to contain the code. Highlight it, then press the [Insert] button.
2. Locate "Class SuperTagging", highlight "CopyTags" below that, then press "Select".
3. Now you'll see two sets of options for tag storage: "Source" and "Target". All the same rules apply to these as to all of the other templates. One additional rule here is that the source and target must be different. Also, if your destination is "BYTE Field in Primary", you'll have the option to "Use ABC RI Code". This will use Relate:Primary.Update() instead of Access: Primary.Update().
4. *New in Super Tagging 6.11* - If you wish to make this operation conditional, enter the Boolean expression in the "Condition" setting. This is wrapped around the entire operation, and NOT applied to each line.
5. If you wish, you may have the target tags wiped before beginning the process, and the source tags wiped after. You can set these to Yes, No, or Ask.

6. If it is set to Ask, it will ask the user whether they want to clear the target and source tags at run time. You can specify the "Title", "Question" and "Icon" for these screens. The "Title" and "Question" settings can be a string (without quotes) or a variable name (prefixed by an exclamation point). Only the "Title" is required.

7. Return to the "Procedure Properties" window, and press the [Files] button. You'll see an entry that says "Copy Tags", with "<ToDo>" under it. Highlight the ToDo line, and press the [Insert] button. Now select the tagged file from the list.

2.14 Tag / Untag / Flip Current Record (Code Templates)

(Code Templates)

All three of these Code templates will be discussed together, because they are very similar. They perform individual tagging operations on the current record in memory, which might be done during a process, or called from a button pressed by the user. The main benefit of these is that they save you the trouble of learning the syntax for calling the tagging support library.

You will be inserting the Code template into an Embed. For example, you might connect it to a "Tag This Record" button. You could also call it as part of the code within a Process procedure. Here's how you do it:

1. Choose an EMBED to contain the code. Highlight it, then press the [Insert] button.
2. Locate "Class SuperTagging", highlight "TagRecord", "UntagRecord" or "FlipRecordTag" below that, then press "Select".
3. Now you'll options for Tag Storage. All the same rules apply to these as to all of the other templates. The only additional rule involves the "BYTE Field in Primary" option. If you choose this, you will see an additional checkbox: "Do not PUT record". If you are tagging the current record in a Form procedure, then the PUT operation would confuse the system. Except in this special situation, this checkbox will normally be off.
4. *New in Super Tagging 6.11* - If you wish to make this operation conditional, enter the Boolean expression in the "Condition" setting. This is wrapped around the entire operation, and NOT applied to each line.
5. Return to the "Procedure Properties" window, and press the [Files] button. You'll see an entry that says "Tag current record", "Untag current record" or "Flip tag", with "<ToDo>" under it. Highlight the ToDo line, and press the [Insert] button. Now select the tagged file from the list.

2.15 Wipe All Tags (Code Template)

(Code Template)

This Code template is useful for clearing the tags from a file. Many of tagging templates give you the option of doing this yourself, but you may still need to call this explicitly.

You will be inserting the Code template into an Embed. For example, you might connect it to a "Wipe Tags" button. Or you might call it in the "End of Procedure" embed after a report. Here's how to set one up:

1. Choose an EMBED to contain the code. Highlight it, then press the [Insert] button.
2. Locate "Class SuperTagging", highlight "WipeTags" below that, then press [Select].
3. Now you'll see the standard Tag Storage prompts. All the same rules apply to these as to all of the other templates.
4. *New in Super Tagging 6.11* - If you wish to make this operation conditional, enter the Boolean expression in the "Condition" setting. This is wrapped around the entire operation, and NOT applied to each line.
5. Return to the "Procedure Properties" window, and press the [Files] button. You'll see an entry that says "Wipe Tags", with "<ToDo>" under it. Highlight the ToDo line, and press the [Insert] button. Now select the tagged file from the list.

2.16 Open and Close TagFile

(Extension Template)

If your procedure performs tagging operations with TagFile_ or TagFilePos_ by calling the tagging library directly, then you may need to include this template in your procedure. (If you have other tagging Extension or Control templates in the procedure, then it is not necessary.)

Once you populate this extension, you must specify the associated data file in the Files window. This controls whether the procedure uses TagFile_ or TagFilePos_.

The extension assumes that you are using a template procedure that includes the "Before Opening Files" and "After Closing Files" embeds. This means that the Source Procedure template will not work with this extension. Instead, you would have to include the following commands at the start of your procedure:

```
Relate:TagFile_.Open                              !or TagFilePos_
Access:TagFile_.UseFile                           !or TagFilePos_
```

At the end of your procedure, you would need this command:

```
Relate:TagFile_.Close                             !or TagFilePos_
```

If your using the Clarion legacy template chain, the appropriate commands will be used.

3 Appendices

3.1 Printing Tagged Records with ReportWriter

If you are using Report Writer, there is a good chance that you would like to process tagged records. This is not too difficult. You have two options. If your tags are stored in a BYTE Field in your data file, you can simply use a filter. Of course this not support multi-user access, as everyone is using the same tag field. If there is normally only one person working with tags, however, then this may be good enough.

In most situations, though, you'll have to use one of the other tagging methods. You should start by understanding the description of the ProcessGetTaggedData extension template, because you will be using much the same method.

If you are using Memory tags, then you will have to use the CopyTags code template to copy them to the corresponding TagFile storage.

The TagFile must be defined in your dictionary. To bring the tag files into your dictionary, import SUPER\LIBSRC\TAGGING\TAGGING.TXD. Your data file must be related to the TagFile, with the data file on the "ONE" side of a "1:MANY" relationship. Once this is done, your report will process the TagFile as the primary file, with your data file as a related file.

Then you must specify a filter so that only corresponding tags are processed. If only one user is tagging records and you are only tagging in a single file in your entire APP, then you could theoretically skip this step. Otherwise, you must set up a filter something like this:

```
TF_:Usr=UserNo AND TF_:Tbl=TagSet
```

"UserNo" is the current user number. If you are not using SuperSecurity, then this will always be zero (in which case you can omit that portion of the filter expression). Otherwise, it will be stored in a variable called UserNo_. You must pass the user number from the program to ReportWriter, or you must have the user enter it in a run-time field.

The TagSet number must match the original number. (If you are using equates, then you must use the literal number in the report filter.)

For example, if you are not using SuperSecurity, and your TagSet is 1, the filter would be:

```
TF_:Usr=0 AND TF_:Tbl=1
```

Finally, you can change the order of the output to match your needs.

3.2 Multi-APP Development using LIBs/DLLs

If you are using the Tagging templates and you want to split your application into multiple LIBs/ DLLs and a single EXE, here's how you do it. This description is based upon the example application in SUPER\EXAMPLES\TAGGING2.

LIB/DLL with the Tagging Code

1. Import the TagFile_ definitions from SUPER\LIBSRC\TAGGING\TAGGING.TXD.
2. Create or open the APP destined to be a DLL/LIB. This must be your base APP in which all of the ABC support code is included.
3. Choose "Application / Properties ..." from the pulldown menu.
4. Blank the "First Procedure" field.
5. Ensure that your "Destination Type" is set to "Dynamic Link Library (.DLL)" or "Library (.LIB)".
6. Click [OK].
7. Press the [Global] button.
8. Press the [Extensions] button.
9. Press the [Insert] button.
10. Select "TagGlobal" under "Class SuperTagging".
11. Press [OK] to exit the "Extensions and Control Templates" window.
12. Turn OFF the "Generate template globals and ABC's as EXTERNAL" check box.
13. Press [OK] to exit the "Global Properties" window.
14. Make the DLL/LIB. (Press the "Lightning" on the button bar, select "Project / Make" from the pulldown, or press Ctrl-M.) You'll find a LIB file in your application directory.
15. If you specified your destination type to "DLL" back in step #5, then you will also have a DLL in your current directory. Remember to include this file when you distribute your APP.

EXE/LIB/DLL without the Tagging Code

1. Create or open the APP that will call the procedures in the APP described above.
2. Press the [Global] button.
3. Press the [Extensions] button.
4. Press the [Insert] button.
5. Select "TagGlobal" under "Class SuperTagging".

6. Press [OK] to exit the "Extensions and Control Templates" window.
7. Turn ON the "Generate template globals and ABC's as EXTERNAL" check box.
8. Press [OK] to exit the "Global Properties" window.
9. Select "Application / Insert Module", then choose "ExternalDLL".
10. Specify your base application name with a .LIB extension as the Name (e.g.: BASE_APP.LIB), then press [OK] to exit the Module Properties window. (Even if your base APP is a DLL, you still use a .LIB extension here.)
11. Make the APP. (Press the "Lightning" on the button bar, select "Project / Make" from the pulldown, or press [Ctrl-M].)
12. Test your APP.

Remember, if you are using DLLs, then all references must be down. For example, you are three APPs: AAA, BBB and CCC. BBB and CCC are DLLs, while AAA is an EXE. AAA calls routines in BBB and CCC. BBB calls routines in CCC. Therefore, CCC must be made first, then BBB, then AAA. If BBB did not call anything in CCC, then you could make BBB before CCC, if desired.

Libraries are different. You can have as many cross references as you wish. LIBs can even call routines in the parent EXE. This is because the linker has all modules (EXE and LIB) together at once, so it can rectify all references, none of which are resolved in a LIB. This contrasts a DLL, which is a standalone module with all of its external references resolved.

3.3 Interface Modification and Translation

We've moved all displayable strings to StAbTag.trn. This file contains equates for messages, titles, buttons and prompts throughout the Super Tagging system.

Feel free to change any or all of these. In most cases, you should copy the file into your application directory. That way new versions of the templates don't overwrite your preferences. For each new APP you create, you can copy your version of the file into that directory.

3.4 Example Programs

There are two example programs provided with the Super Tagging templates. These can be found in SUPER\EXAMPLES\TAGGING1 and 2. The only difference between these is that the second demonstrates how to place the tagging support module into a DLL. These examples are referenced repeatedly within this documentation.

The TAGGING1 directory contains TEST.DCT, TEST.APP and EMPLOYEE.TPS. The TAGGING2 directory includes an additional SUPPORT.APP.

The example contains five browse procedures. Three of these are virtually identical, except that they use different tag storage methods. They can also pass the tags in a circle using the CopyTags code template. These procedures also demonstrate various control and code templates.

There is another Browse that utilizes Window-style marking. There is also a browse that implements icons in the tag column of the browse box and on the tagging buttons. It also demonstrates the SaveTags and LoadTags templates. Both of these browses save their tags in memory.

Finally, there are four reports that print tagged records. Three of these use the ProcessTagFilter extension, which applies a filter to a regular Procedure or Report. The last one uses the ProcessGetTaggedData extension, which is much faster by accessing the tag file directly. It also allows you to control the order of the output.

3.5 API Reference - Tagging

<u>Tagging API for Memory (Integer Primary Keys)</u>

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_ )`

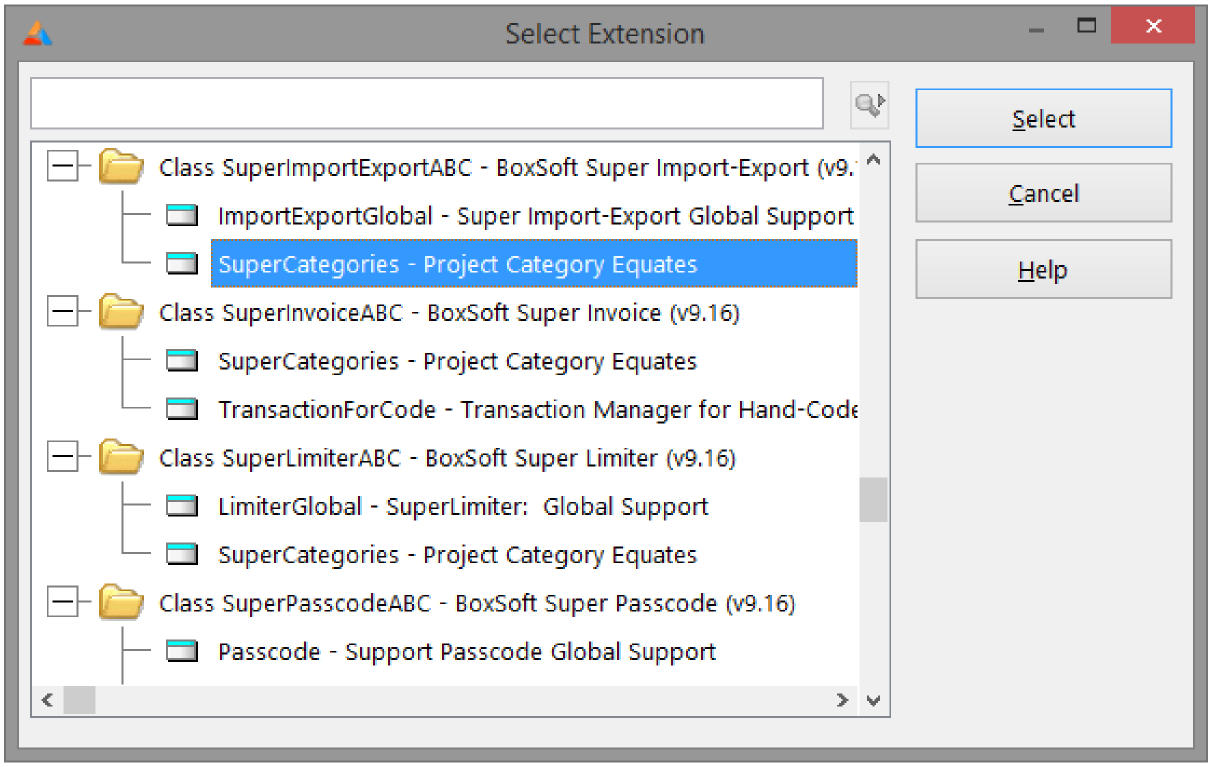
This procedure is used to set the value of the current "user". If you're using the BoxSoft Super Security templates, then this is handled for you automatically. If you're using another method for security, you need to call this procedure whenever your user changes. Usually this will be only at the start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

3.6 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.

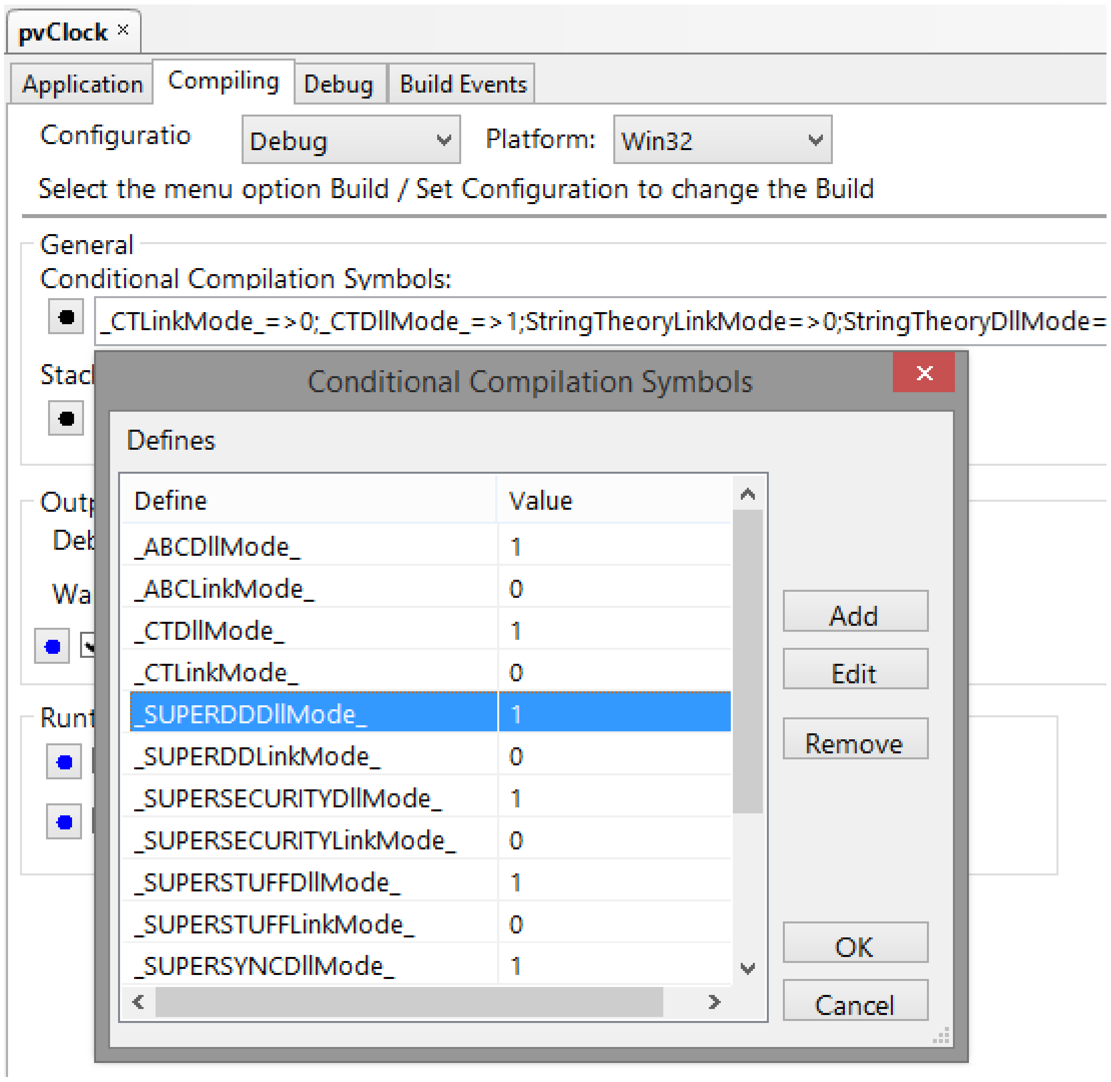
To make this inclusion more selective, as of our 7.0 versions we changed to use various other switches. Some of these are product related (e.g. Super QBE uses _SuperQBELinkMode_ and _SuperQBEDllMode_), while others are associated with one of our shared base classes (e.g. Drag & Drop uses _SuperDDLinkMode_ 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.



Class SuperImportExportABC - BoxSoft Super Import-Export (v9.
ImportExportGlobal - Super Import-Export Global Support
SuperCategories - Project Category Equates
Class SuperInvoiceABC - BoxSoft Super Invoice (v9.16)
SuperCategories - Project Category Equates
TransactionForCode - Transaction Manager for Hand-Code
Class SuperLimiterABC - BoxSoft Super Limiter (v9.16)
LimiterGlobal - SuperLimiter: Global Support
SuperCategories - Project Category Equates
Class SuperPasscodeABC - BoxSoft Super Passcode (v9.16)
Passcode - Support Passcode Global Support

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:


pvClock ×

Application | Compiling | Debug | Build Events

Configuratio Debug Platform: Win32

Select the menu option Build / Set Configuration to change the Build

General

Conditional Compilation Symbols:

CTLinkMode=>0;_CTDllMode_=>1;StringTheoryLinkMode=>0;StringTheoryDllMode=

Conditional Compilation Symbols

Defines

Define	Value
ABCDllMode	1
ABCLinkMode	0
CTDllMode	1
CTLinkMode	0
SUPERDDDllMode	1
SUPERDDLinkMode	0
SUPERSECURITYDllMode	1
SUPERSECURITYLinkMode	0
SUPERSTUFFDllMode	1
SUPERSTUFFLinkMode	0
SUPERSYNCDllMode	1

3.7 Troubleshooting

Problem:

When you run your application, the system is reporting that the TagFile is bad.

Solution:

Because the TagFile is usually a scratch file (unless you are using it to store long term tag selections), just delete and let the system recreate it. The default names for these files are TAGFILE_.TPS and TAGFILEP.TPS.

Problem:

The system complains that it cannot find TagSet_ and fields with the prefix TS_.

Solution:

Check that you have TagFile_, TagFilePos_ and TagSet_ in your dictionary. If necessary, import them from SUPER\LIBSRC\TAGGING\TAGGING.TXD.

3.8 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

3.9 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