

**BoxSoft**  
Corporation

## **Super Security - Documentation**

Copyright © 1989-2014, BoxSoft Corporation, All Rights Reserved.

# Table of Contents

Foreword	0
<b>Part I Getting Started</b>	<b>4</b>
1 Introduction.....	4
2 RTFM Warning!!!.....	7
3 Installation.....	8
4 What is Security?.....	12
5 Upgrading from Earlier Versions.....	17
<b>Part II Template Usage</b>	<b>23</b>
1 Adding Security to your Applications.....	23
2 Support Files and Procedures.....	24
3 Global Extension Template.....	27
4 Procedure Security.....	35
5 BrowseBox Update Security.....	37
6 Form Update Security.....	40
7 Protecting Controls on a Window.....	43
8 Protecting Controls on a Report.....	44
9 Auditing Access to a Procedure.....	45
10 Creating an Audit Trail Entry.....	46
11 Auditing Field Edits During a Change Operation.....	47
12 Button for Run-time Security Maintenance.....	48
13 Conditional Code.....	49
14 Running a Program to "Check for Previous Logon".....	50
<b>Part III Appendices</b>	<b>52</b>
1 Support Programs - DoorEdit and UserEdit.....	52
2 Run-time Security Implementation.....	56
3 Multi-APP Development using LIBs/DLLs.....	57
4 SQL and Super Security.....	59
5 Disabling Security.....	60
6 Interface Modification and Translation.....	61
7 Example Programs.....	62
8 API Reference.....	64
9 Project Defines.....	75
10 Troubleshooting.....	77
11 Contacting Technical Support.....	79

---

12 License Agreement.....	80
<b>Index</b>	<b>81</b>

# 1 Getting Started

## 1.1 Introduction

The Super Security templates enable you to protect your applications against unauthorized users. They work with Clarion for Windows 5.5 and greater, using both the legacy and ABC templates.

With Super Security, you can assign access to users by levels or doors. (This is not the same as our Super Passcode templates, which protect against software piracy.) These restrictions can be defined during development and/or at run-time. You can also record an audit trail of system usage.

You can specify whether a logon window is generated, and whether it asks for the logon at start-up or at the first security check. You can also have the system check for a previous logon in a calling program. There is a backdoor username for your system support needs.

You can also allow a "manager override" on a global or local basis. The override duration can be one-time or permanent. This override feature is configurable for each access point.

We've included DoorEdit (for you to maintain your doors) and UserEdit (for your users to update their user records) to help you implement and maintain your security system.

All messages and text in the security system can be modified to suit your needs. There are also both global and local "access denied" messages that can be modified, if you so desire.

You can also optionally provide "View" support when the user doesn't have access to change the record.

There are a variety of templates included in this package, to aid in implementing security:

**SecurityGlobal** - This Extension template is populated into your global area to provide the basic security support.

**ProcedureSecurity** - This Extension template restricts access to a procedure. It checks for valid rights immediately upon being called, and returns if the user does not have access. Of course, you can optionally allow manager overrides.

**BrowseUpdateSecurity** - This Extension template restricts access to BrowseBox operations like Insert, Change and Delete. You can control the Level or Door, the "Access Denied" message, and the Manager override ability for each operation. Unavailable operations can have their button disabled, hidden, or available with an "Access Denied" message. There is a control template called BrowseUpdateSecurityC that can also be used.

**FormUpdateSecurity** - This Extension template is essentially the same as the BrowseUpdateSecurity. The only difference is that it is implemented on the Form side, instead of with the BrowseBox.

**WindowControlSecurity** - This Extension template restricts access to window controls. For example, it can be used to hide the salary field when inappropriate. Any or all of the window controls can be affected, with each having its own level/door and disable/hide

setting. You can also restrict access to ranges of controls (i.e.: HIDE(?Emp:Salary: Prompt, ?Emp:Salary)).

**ReportControlSecurity** - This Extension template restricts the printing of report controls. For example, it can be used to hide the salary field when inappropriate. Any or all of the report controls can be affected, with each having its own level/door setting. You can also restrict access to ranges of controls.

**ProcedureAudit** - This Extension template records an entry in the audit trail whenever the procedure is entered. It records the date, time, user, primary key and description to help to track the user's activity.

**ChangeAudit** - This Extension template is used in conjunction with the FormUpdateSecurity template. It remembers the changes made to all fields during a ChangeRecord operation.

**AuditAccess** - This Code template lets you manually add an entry to the audit log from within one of your own embed points. (You can also call the API functions directly, instead.)

**InvokeRuntimeSecMaint** - This Control template is populated as a button on your frame's toolbar. The user can hit this button to invoke runtime security maintenance. If the user doesn't have the proper access, the button will be hidden.

**RunSecurity** - This Code template is used to issue the RUN() command with the "Check for Previous Logon" information. Consequently, the called program can skip the logon window. (If the child program is called directly, it will still require a logon.)

**ConditionalCode** - This Code template is used to execute some code, depending on the user's access rights.

There are also various support procedures that can be called directly, like Security.Logon and Security.CheckAccess.

### **ABC and Legacy Template Chains**

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

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

For more information, see the following topics:

Adding SuperSecurity to your Application  
Support Files and Procedures

- Security Global Extension
- Procedure Security
- BrowseBox Update Security
- Form Update Security
- Protecting Controls on a Window
- Protecting Controls on a Report
- Auditing Access to a Procedure
- Creating an Audit Trail Entry
- Auditing Field Edits During a Change Operation
- Button for Run-time Security Maintenance
- Conditional Code
- Running a Program to "Check for Previous Logon"
- Support Programs - DoorEdit and UserEdit
- Run-time Security
- Multi-APP Development using LIBS/DLLs
- SQL and Super Security
- Interface Modification and Translation
- Example Programs
- API Reference
- Troubleshooting
- Contacting Technical Support
- License Agreement

## 1.2 RTFM Warning!!!

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

## 1.3 Installation

### Installation Directory Structure

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

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

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

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

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



```

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

```

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

```

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

```

For example, you can use a tool like the indispensable Beyond Compare ([www.scootersoftware.com](http://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.

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

### 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 is Security?

The Security features enable you to protect your program from unauthorized use. This can be as simple as forcing the user to logon when they start the program. Then you can optionally protect just those areas of your program that need it. Usually you won't need to protect everything. The security facility allows you to easily add security without constantly forcing you to do too much work. You can even allow your users to add security at run-time.

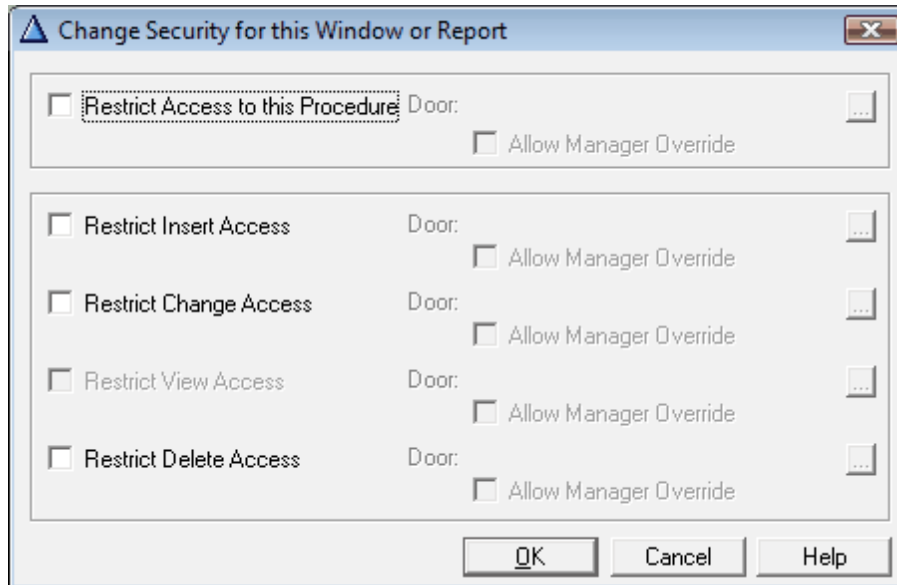
You have complete control of messages, manager overrides, delayed logon (until the first checkpoint), field-level security, and much more.

Please continue to give us the feedback. We know that no tool is perfect, and we want ours to suits your needs as best as it can.

### Users and User Groups

Each person accessing your system should be stored in the built-in database as a User. You can also define as many User Groups as necessary. Each user can be in any number of groups. If a user doesn't have personal access to a procedure, but they are in a group that does, then they are granted access implicitly. You can also deny a user's access to a particular door, regardless of the user's group status.

### Run-time Security



Sometimes you won't know what your users want to protect in their systems (or maybe you don't have time to implement all of the doors yourself). In this situation, you can globally enable run-time security maintenance. Users with sufficient rights will be able to add, change and remove restrictions from procedures and forms at their own discretion.

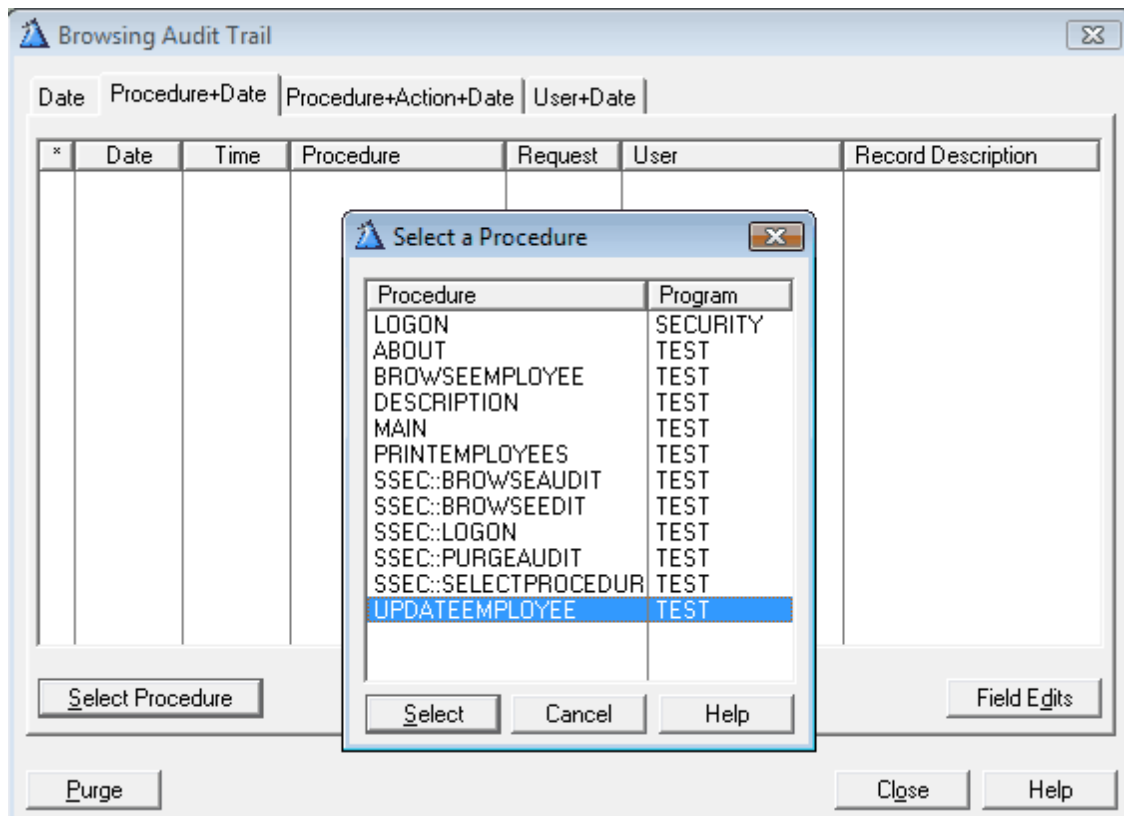
One of the best features is that they can't modify any access checkpoints that you implemented

during development. This enables you do provide some "hard-coded" security for key areas, and let them twiddle with the rest.

When the maintenance mode is instigated, it applies only to the currently visible procedure (or report that was just started). To change the run-time security for a particular procedure, you must be running that procedure when you enter the maintenance mode.

By the way, run-time security never applies to your program's frame.

## Auditing



Sometimes you need to record who's doing what. This is where the auditing feature comes in handy. You can track important procedures for user access. The system records the date, time, user, and an optional primary key value and description. If you're auditing an update form, it will also record the action (insert, change, etc.), and whether it was canceled. You can even track the modification of fields during a ChangeRecord operation. Finally, the system will record whether the user was denied access to the audited procedure.

To prevent the audit file from growing too quickly, auditing is implemented on an as-needed basis via a number of extension and code templates. There is a browse provided to peruse the audit log, as well as a purge facility for remove old entries.

## Security Types

There are two different types of security provided: Levels and Doors.

### ***What are Levels?***

With levels, the user is assigned his or her level. The programmer specifies which operations and controls are protected at which levels. The system administrator can control the various users' levels, but not the levels required for each procedure.

It's up to the programmer (that's you) to organize the various protected procedures into "authorization circles". You can leave this to your user to assign with run-time security, but it's still not very flexible.

For example, the clerk can access certain things; the manager can access everything of the clerk's, and a little more; the president can access everything the manager can, and a little more; etc. It is not possible to allow the clerk to access something separately from the manager.

Our levels example (found in both SUPER\EXAMPLES\SECUR\_L1 and SECUR\_L2) is an extremely simple employee update system. There are two levels:

- |                    |  |
|--------------------|--|
| <b>1 - Clerk</b>   | Add and Change Employees   |
| <b>2 - Manager</b> | Add and Change Employees, and also Delete Employees, Print Employees, Access Salaries, Update Security |

You'll notice that level 2 is more "sensitive" or "important" than level 1. A clerk with level 1 would be able add and update employees, but could not delete them or access their salaries. With level 2 you could also delete, plus perform many other procedures.

In fact, our example program does not even bother to assign level 1 access to any procedure. This is because all individuals who have access to the system will be at least level 1. Protection within the program starts at level 2.

In most cases, it's suggested that you refrain from using Levels, and instead to strongly consider using Doors.

### ***What are Doors?***

In most tougher situations, you will probably want to use Doors. You assign a different door to each protected procedure (or the same door to a bunch of procedures). The administrator can assign any combination of doors to each user. This is a little more work for the administrator, but that's what they get with the added flexibility.

For example, the first user can access procedures A, B and C; the second user can access procedures A, C and D; the third user can access procedures B, C and E; etc.

<b>User 1</b>	A	B	C		
<b>User 2</b>	A		C	D	
<b>User 3</b>		B	C		E

It does not matter what door number you specify for a particular procedure. In fact, you can specify the same door number for more than one procedure. For example, the "manager" door could control access to many different procedures associated only with the manager's duties. Each door is associated with a description, to make it easy to assign access to various users.

This is a very flexible approach. If your users are paranoid, you could assign a different door to every procedure in your application. Then the system administrator could control which procedures each user could access. Of course, this would be overkill.

Alternatively, you could go with a minimalist approach. Let's pretend that there are six different operations that are performed by your system: Entering Sales, Cashing Out, Updating Customers, Printing Reports, and Managing Inventory, and Performing Miscellaneous Operations. There might be 100 procedures in your application, but they all fit within one of these six areas. You could create six doors, and assign one of these doors to each of the corresponding procedures.

Neither of these systems would be wrong, but neither of them is realistic; you would likely have your system somewhere between the two. You probably have five to ten general areas of functionality in your program. You should create a door for each of these areas. If there are any extra operations that don't fit into these main areas, they could get their own doors. If you decide that certain specific procedures within the main areas need to be protected separately, then these would get their own doors.

Remember, you don't need to protect your entire application. There are probably many procedures that can be accessed by anyone using the program. Why protect those areas when everyone will need to be granted individual access to them.

Of course, you could leave all this up to the administrator, and let them assign all of their doors with the Run-time security.

Keep in mind that a procedure called by a protected procedure cannot be reached if the user does not have access to the calling procedure. Of course, the called procedure may need higher access, in which case it would need its own door. For example, some people are allowed to edit employee records, so the employee browse is assigned the eD\_ViewEmployee, eD\_UpdateEmployee and eD\_DoorsEmployee doors. However, only the manager is allowed to change the salary information. Therefore the salary update window would use the eD\_AccessSalary door.

Our doors example (found in both SUPER\EXAMPLES\SECUR\_D1 and SECUR\_D2) is an extremely simple employee update system. Because doors provide more flexibility, we have assigned a different door to each operation that we wish to protect.

```
eD_DelEmp      EQUATE(1)  !Delete Employee
eD_Salary     EQUATE(2)  !Access Salary
```

```
eD_EmpReport  EQUATE(3)  !Print Employee Report
eD_Security    EQUATE(4)  !Edit User Security
eD_ChgEmp      EQUATE(5)  !Update Employee
```

These are all of the operations that are grouped into "level 2" in the levels examples.

Each door must be assigned to a door group.



## 1.5 Upgrading from Earlier Versions

### Super Security 6.04 to 6.05

If you're using the legacy chain, you must import CLARION\SUPER\_SRC\_CLA\SECURITY\LOGON.TXA into your base APP. Once you do that, you can make any desired changes to the window directly in your APP, instead of in STCLSEC.TRN.

### Super Security 6.03 to 6.04

If you want to store the Security files in an SQL back-end, then you need to re-import LOGON.TXA to get the new SSEC::Logon procedure. Alternatively, you can manually edit SSEC::Logon and replace ST::FindUser with the following code:

```
ST::FindUser PROCEDURE(STRING p_Name) !,BYTE
L::RetVal          BYTE,AUTO
L::Manage:UserView  ViewManager
L::UserView         VIEW(SSEC::User).
CODE
L::Manage:UserView.Init(L::UserView, Relate:SSEC::User)
L::Manage:UserView.AddSortOrder
L::Manage:UserView.SetFilter(
    'UPPER(LEFT(CLIP(SUser_:FirstName) & ' ' & SUser_:LastName)) = ' ' ' |
    & ST::Quote(UPPER(CLIP(p_Name))) & ' ' AND SUser_:GroupFlag=0')
L::Manage:UserView.Reset
L::RetVal = CHOOSE(~L::Manage:UserView.Next())
IF L::RetVal                                     !MH 10/19/03 SQL
    REGET(L::UserView, POSITION(L::UserView))     !MH 10/19/03 SQL
END!IF                                           !MH 10/19/03 SQL
L::Manage:UserView.Close
L::Manage:UserView.Kill
IF L::RetVal                                     !MH 12/18/03 SQL
    WATCH(SSEC::User)                           !MH 12/18/03 SQL
    L::RetVal = CHOOSE(~Access:SSEC::User.Fetch(SUser_:NoKey)) !MH 12/18/03 SQL
END!IF                                           !MH 12/18/03 SQL
RETURN L::RetVal
```

### Super Security 5.x to 6.x

The primary changes involve support for the new thread handling in Clarion 6. There are a number of class properties that were formerly available, which must be accessed now via wrapper methods. In all cases, the access methods are called **Set\_Variable(NewValue)** and **Get\_Variable()**. For example, **BackdoorUsername** was directly accessible before, but now you must set it with **Set\_BackdoorUsername('NewValue')** and retrieve the current value with **Get\_BackdoorUsername()**. The variables affected are:

UserNo_	Security.Set_UserNo(val) Security.Get_UserNo()
UserName_	Security.Set_UserName(val) Security.Get_UserName()
Security.AutoFillProgram	Security.Set_AutoFillProgram(val) Security.Get_AutoFillProgram()

Security.BackdoorUsername	Security.Set_BackdoorUsername(val) Security.Get_BackdoorUsername()
Security.PrevLogonINI	Security.Set_PrevLogonINI(val) Security.Get_PrevLogonINI()
Security.PasswordEncrypt	Security.Set_PasswordEncrypt(val) Security.Get_PasswordEncrypt()
Security.PasswordEncrypt	Security.Set_PasswordEncrypt(val) Security.Get_PasswordEncrypt()

If you try to compile your programs without addressing these (**UserNo\_** and **UserName\_** in particular), then you'll probably get some compiler errors.

We've made the necessary changes to **SSEC::Logon** in LOGON.TXA. You can make these same changes yourself, but the best approach is to import the new LOGON.TXA to overwrite your existing **SSEC::Logon** procedure. Of course, if you've made changes to your own procedure since importing it, then you would have to make your own changes again. We leave it up to you to take the best approach for your circumstances. If you decide to make the manual changes, here are some example to help you out:

```

Old:  GETINI('SuperSecurity', p_Security.AutoFillProgram)
New:  GETINI('SuperSecurity', p_Security.Get_AutoFillProgram())

Old:  UserNo_ = -1
New:  p_Security.Set_UserNo(-1)

Old:  UserName_ = p_Security.Translate('SSEC::BackdoorDisplayName')
New:  p_Security.Set_UserName( |
      p_Security.Translate('SSEC::BackdoorDisplayName'))

Old:  p_Security.AddCall('Security', 'Logon',,, UserNo_, UserName_)
New:  p_Security.AddCall('Security', 'Logon',,, |
      p_Security.Get_UserNo(), |
      p_Security.Get_UserName())

```

### **Super Security 4.x to 5.x**

The primary changes involve the new file SSEC::PwdLog, changes to the file SSEC::User, and a new procedure called SSEC::Logon that you must import into your APP. You should also create a new UserEdit, as the SSEC::UpdateUser procedure has been updated.

#### **SSEC::PwdLog**

There is a new file called SSEC::PwdLog. You can simply copy & paste this from C:\C55\SUPER\LIBSRC\SECURITY\SECURITY.DCT into your DCT. Set up a MANY:1 relationship with SSEC::User (i.e. many PwdLog records for one User). Set the the referential integrity constraints to Cascade for both Update and Delete.

#### **SSEC::User**

The SSEC::User has also been changed. There are many ways you could perform this conversion, and here's one good approach:

1. Before you change anything, generate the source code for your application, so you get a

source-code version of your old file structure. (Make sure that you save this somewhere.)

2. Examine the new structure of SSEC::User in C:\C55\SUPER\LIBSRC\SECURITY\SECURITY.DCT. In particular, the Password field has been changed to 20 characters, and we've added the following fields: PasswordSize, PasswordMaxAge, PasswordDate, PasswordTime, LogonDate, LogonTime, Failures, and Locked.
3. Make the equivalent changes to SSEC::User in your own dictionary. TIP: This is a great time to use a multi-clip utility like ClipMate ([www.thornsoft.com](http://www.thornsoft.com)), so that you can copy all of the fields from one dictionary, and then power-paste them all into the next.
4. Generate the source code for your application, so you get a source-code version of your new file structure.
5. Copy SEC\_CONV.PRJ and SEC\_CONV.CLW from C:\C55\SUPER\LIBSRC\SECURITY\CONVERT50 into your application directory.
6. Load and edit SEC\_CONV.PRJ. If you're using a different database driver, you'll have to change it here. Also, if you're using C5 (rather than 5.5), then you have to remove C55TPS%X%%L%.LIB from the "Library, object, and resource files", and then add "TopSpeed" to the "Database driver libraries".
6. Load and edit SEC\_CONV.CLW:
  - a) Change the initial value of UserFilename to match your own filename.
  - b) Edit the OLD\_User structure to match your old structure. Keep the prefix as "OLD" in both the FILE and KEY lines.
  - c) Edit the SSEC::User structure to match your current file structure. (If you've made the changes to your file to exactly match the sample SECURITY.DCT, then you won't need to make any changes.)
7. Compile and test the program.

**NOTE:** This conversion program reads the contents of the existing user file into a queue, removes the file, then writes the new file in its place. Therefore, it's strongly recommended that you make a backup copy of your data file before attempting this conversion.

### SSEC : : Logon

We've got all of the hard-coded windows into regular window procedures, to make SuperSecurity more compatible with the likes of Clarionet. The last was the Logon window. You need to import LOGON.TXA from C:\C55\SUPER\LIBSRC\SECURITY into your base APP (that's the one where your files are generated). Also, if you are not using global maps, then mark the newly-imported procedure as "Declare Globally" in its procedure properties.

### USEREDIT . APP

The procedure SSEC::UpdateUser has been updated in both USERED\_L.TXA (for levels) and USERED\_D.TXA (for Doors). You should create a new UserEdit app for your system.

If you've made significant changes to your own UserEdit, then you may want to create a second UserEdit for comparison (or look at the existing example apps), and then manually apply the changes to your app. There are two primary changes to this procedure:

- All the new fields in the SSEC::User file have been added to the window. Some of these are user-editable, while others are display strings.
- If you're using the large Password size, ensure that the field is bigger, and that the entry control's picture matches the field size (e.g. @S20).
- In two places in the embeds, the call to the Security.EncryptPassword has changed. In `ThisWindow.Init`, it looks like this:

```
SUser_:Password = Security.EncryptPassword( SUser_:Password, Security.PasswordSize())
```

- And in `ThisWindow.TakeCompleted`, add a new line first and change the existing line, so that it looks like this:

```
SUser_:PasswordSize = LEN(CLIP(SUser_:Password))
SUser_:Password      = Security.EncryptPassword( UPPER(SUser_:Password), SUser_:
PasswordSize )
```

#### DOOREDIT.APP

There is one new global data embed in DOOREDIT. You can either recreate the APP, or add the following line to the global data area:

```
SSEC::DefaultMaxPwdAge EQUATE(0)
```

### Super Security 2.x to 4.x

If you are converting an application to the new ABC templates, the BoxSoft rules are present in Clarion's conversion wizard to help you to perform the task.

### Super Security 1.9x to 2.x

We've made some significant changes between SuperSecurity 1.9x and 2.0. Here's a basic overview, followed by step-by-step conversion instructions. Some of the elements in the list are abbreviated, because they are explained in more detail elsewhere.

Here are the major changes:

- All security files must be imported into your dictionary. See Support Files and Procedures for more information. Also, the files have been changed, so you must run them through a conversion program (see the following conversion instructions).
- There are some required and some optional procedures to be imported into your application. See Support Files and Procedures for more information.
- The support procedures and functions have been converted to an OOP class. This changes the name of the source procedures, in case you are calling them in your own source code. See the API Reference for more information.

- UserEdit and DoorEdit have changed significantly. You must recreate these applications. See Support Programs for more information.
- Many of the global settings for strings have been moved out of the global extension, and into C:\C55\LIBSRC\STABSEC.TRN. See Interface Modification and Translation for more information.

### Step-by-Step

Converting your APPs from the old security to the new can get a little involved, but if you follow these instructions closely, you should manage quite well.

1. Backup your APPs. Especially USEREDIT and DOOREDIT, as they will be replaced. You will likely refer back to these APPs for your various template settings.
2. Import the security files from SUPER\LIBSRC\SECURITY\SECURITY.TXD into your dictionary. Do not remove any of these files from your dictionary. If your dictionary contained the old security files, leave them there for now!
- 3a. Create a new APP called UserEdit in your application directory. Tell it to use your dictionary. Import either SUPER\LIBSRC\SECURITY\USERED\_L.TXA or USERED\_D.TXA (depending on whether you are using Levels or Doors) into the new APP. When it complains about the dictionary error, ignore it and wait for the X to appear! Then hit Close. Delete empty modules, then renumber modules. (Both of these are under the Application menu.)
- 3b. Go into the Global settings, then to Extensions. Now change these settings to match your setup. You may want to have it check for previous logon, if you are calling UserEdit from an APP with its own logon.
- 3c. Go into the Global Embeds and change the "32000" in the Procedure Setup to match your security maintenance access level/group.
4. If you are using Doors, do the same thing for DoorEdit (except that there is only one import file called DOOREDIT.TXA). DoorEdit doesn't contain the security extension, so you can skip steps b and c.
5. If your applications directly access the security files, these accesses must be changed manually. If you had previously imported the old security files into your dictionary and created procedures using them, then you will probably have to recreate these procedures.

However, you can leave the old security files in the dictionary until you've finished getting the new files in place and the manual references in your procedures changed. Then you can delete the old files from the dictionary and test your work.

6. To convert your old security file data, copy SEC\_CONV.APP and SEC\_CONV.DCT from SUPER\LIBSRC\SECURITY\SEC\_CONV into your application directory.

Then verify that the file definitions for User\_, Door\_ and Access\_ in the dictionary match your old security files. You can verify this by trying to Browse them from the dictionary. If it complains about an invalid file, or wants to convert the file, then you're not quite there.

Once that's done, you should be able to compile the APP. If you are using Levels rather than

Doors, then modify the Convert source procedure. Remove the calls to MakeDoorGroup, ConvertDoors and ConvertAccess. Distribute the compiled EXE to your users to convert their security files (or you can convert it to a DLL, if you wish).

## 2 Template Usage

### 2.1 Adding Security to your Applications

The SuperSecurity templates are implemented via a orchestration of imported TXDs and TXAs, templates and functions and classes. The templates should handle all interfacing with the Class, so you should rarely (if ever) have to call them yourself. If you are adventurous, though, you can certainly give it a try. (For more information on this, see the API Reference.)

To add the SuperSecurity templates to your application, you must begin by importing one TXD into your dictionary, and optionally a couple of TXAs. Then you add the Global Security Extension template, followed by any number of other templates throughout your application.

For more information, see:

- Support Files and Procedures
- Security Global Extension
- Procedure Security
- BrowseBox Update Security
- Form Update Security
- Protecting Controls on a Window
- Protecting Controls on a Report
- Auditing Access to a Procedure
- Creating an Audit Trail Entry
- Auditing Field Edits During a Change Operation
- Button for Run-time Security Maintenance
- Conditional Code
- Running a Program to "Check for Previous Logon"

## 2.2 Support Files and Procedures

### *(TXDs and TXAs)*

In SuperSecurity 2.0, we decided to always place the security files in the dictionary. It gives us many benefits, and clears up much confusion regarding the settings for the global extension template. We've also developed several of the support procedures that you will import into your application from TXA files. (Always remember to use "Import Text" to pull these TXAs into your APPs.)

### Importing the Security Files

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

**SSEC::User** This is the database of users. You can add your own fields to this file, if you wish. If, instead, you have your own user file, you must ensure that there is a corresponding record in this file for each user in your file.

You'll also find the "User Groups" stored within this file. The GroupFlag field indicates whether the record represents a group, rather than a user.

With the release of version 5.0, we added new fields for handling password expiration.

**SSEC::UserInGroup** This file "Joins" the SSEC::User file and SSEC::UserGroup alias. This enables us to have a many-to-many relationship between users and groups. (i.e.: One group can have many users, and one user can be in many groups.)

**SSEC::UserGroup** This is an ALIAS of the SSEC::User file. It simplifies APP development when working with Users and User Groups (which are stored in the SSEC::User file). The SSEC::UserInGroup file is used to join this alias and the original SSEC::User file.

**SSEC::PwdLog** This contains the password history for each user.

**SSEC::Door** This file contains the various access points referenced throughout your program. These doors are displayed when editing users.

This file contains a field called "Freeze". By default, all records entered within DoorEdit will be frozen, while those entered with run-time security will not. If a record is frozen, it cannot be updated within the run-time security maintenance windows.

If you are using Levels rather than Doors, you still need this file for the API code to compile properly.

**SSEC::DoorGroup** This file contains groupings for the doors. It helps keep the doors organized when there are too many of them to maintain in a single group.

This file contains a field called "Freeze". By default, all records entered within DoorEdit will be frozen, while those entered with run-time security will not. If a record is frozen, it cannot be updated within the run-time security maintenance windows.

**SSEC::Access** This file contains a record of which users have access to which doors. It also



contains a DenyFlag field, which indicates that a particular user does *not* have access to the door, regardless of their User Group affiliations.

<b>SSEC::Program</b>	This contains a list of all programs in your system. If your system contains only one APP, then there will be one or two entries in this file: the name of your APP and "SECURITY" (if you are auditing logons).
<b>SSEC::Procedure</b>	This contains a list of procedures in an APP, plus a pointer back to a SSEC::Program record. This is where the run-time security information is stored.
<b>SSEC::File</b>	This contains a list of the data files in your system.
<b>SSEC::Field</b>	This contains a list of fields in your data files. Each record contains a pointer back to SSEC::File.
<b>SSEC::Call</b>	This file contains the audit trail records. There are pointers to SSEC::Procedure (and implicitly SSEC::Program), SSEC::User, and SSEC::File. There are also fields optionally containing the Date, Time, Request (e.g.: InsertRecord), Primary Key, Description, whether the access was denied, and whether the update operation was cancelled.
<b>SSEC::Edit</b>	This file contains the field update information when you are using the ChangeAudit extension template. It contains a pointer to SSEC::Call and SSEC::Field, plus the old and new values for the field.

All of these files are stored in a single "megafile", using the TopSpeed file driver. The filename itself is SSECUR\_TPS. They are also encrypted with an owner name of "BOXSOFT". If you want to change this, be sure to change it in all files.

If you decide to use the Primary Key support with the Auditing feature, then you may want to use the PrimaryKey field in SSEC::Call. The primary key in your own data file must contain a single numeric field for this to work. Set up the relationship with the SSEC::Call file on the MANY side of the relationship with "No Key". Hence, a one-way look-up relationship.

Do *not* remove any of these data files from your dictionary. Also, do not remove any of the fields, or modify any of the pre-existing keys and relationships. You *are* allowed to add any additional fields to these files, if it suits your development needs.

**NOTE 1:** You will be adding Doors and Door Groups to this file during development. You will also probably be create test records in the various other security files. Before sending the system to your users, you must eliminate all of test data from SSECUR\_TPS. Probably the easiest method is to use the *TopSpeed Database Copy Utility* to copy SSEC::Door and SSEC::DoorGroup to a fresh file in another directory. You may also wish to provide a few sample users to help the administrator get off on the right foot.

**NOTE 2:** If you're using SQL, then you may need to change the NAME attribute for some of the keys, so that they are unique on your server. (With many SQL implementations, you cannot have two tables with the same key name.) The main culprit key is "NoKey". The easiest thing is to use "*TableName\_KeyName*". You may also have to do this for some of your fields, if they happen to use a reserved word on your SQL implementation (or you can wrap the field name in quotes).

**NEW in Super Security 6.04:** We added a new global embed, so that you can assign runtime filenames for the security files without going to the trouble of overriding the Security class. Look for "[SuperSecurity] Prepare Security Filenames".

### **Importing the Logon Procedure**

Due to the popularity of ClarioNet et al., it became necessary to change the Logon into a regular, generated Window procedure (as of SuperSecurity 5.0). All existing and new systems require the new SSEC::Logon procedure. The name of the import file is LOGON.TXA. Many of the new features in version 5 are supported by code in this procedure. If you want to tweak the logon procedure, this is the best place to do it.

If you are creating a multi-APP project, import it into the APP that contains the Security Global extension marked as "Internal" (i.e.: your "base" support APP). This procedure is automatically called by the old *Security.Logon* method.

For the Clarion/Legacy chain, this was implemented in Super Security 6.05.

### **Importing the Run-time Security Maintenance Procedures**

If you wish to use the Run-time Security feature, you *must* import the support procedure(s) into your application. If you are creating a multi-APP project, import it into the APP that contains the Security Global extension marked as "Internal" (i.e.: your "base" support APP).

Depending on whether you are using Levels or Doors, import either RUNTIMEL.TXA or RUNTIMED.TXA from SUPER\LIBSRC\SECURITY. If you are using Levels, you will get only one procedure. If, however, you are using Doors, then you will get four procedures.

You can modify the appearance of any of these procedure to match your style, language, etc. If you want to modify the original, you'll find the source APPs in directories below the TXAs. When you are done modifying the APPs, be sure to do a "Selective Export" of only those procedure whose names begin with "SSEC::".

If you are using doors, the run-time maintenance procedures can be used to add and modify doors and door groups. However, they cannot modify the doors and door groups created during development by DoorEdit. (This is restriction is controlled by a field called "Freeze" in each of the two files. The Freeze values may be modified with DoorEdit.)

### **Importing the Audit Browsing Procedures**

If you want to browse the audit trail, you can import these procedures into your APP. Remember that you'll be calling the lead procedure yourself, so import the APP wherever it makes the most sense. Of course, the APP must share your dictionary containing the security files.

The name of the import file is SUPER\LIBSRC\SECURITY\AUDIT.TXA. Once you've pulled it into your APP, just call SSEC::BrowseAudit like any other procedure. This lead procedure also contains the ProcedureSecurity extension template, so you should change the door setting to match your system (or you could remove the template and set it at run-time).

As was mentioned above, you can modify the procedures once they are in your APP, or you can change the original AUDIT.APP and selectively re-export the original TXA procedures.

## 2.3 Global Extension Template

### *(Extension Template)*

To use the Security features, you must add the global support to your application. Just press [Global], [Extensions] then [Insert]. Look for "SecurityGlobal" under the SuperSecurity section.

Many of the prompts may not be visible, depending on your application's status. You'll see the most prompts if this is a standalone EXE (not requires other APPs of your own making). There are slightly fewer if your APP acts as the "base" app (i.e. contains the file definitions and class libraries) in a multi-APP system. There are far fewer options if this is a dependant EXE (i.e. the APP creates the EXE portion of a multi-APP system). You'll see the fewest options if this is a "middle" DLL (i.e. it's not the base APP or the EXE app).

The common settings that you specify in the base APP will be saved in a file called STABSEC.INI. The rest of the APPs will automatically share the same settings.

### **General Tab**

These settings control the availability of many prompts:

- Generate template globals and ABC's as EXTERNAL in "Global Properties"
- Destination (EXE or DLL) in "Application Properties"

Support Logon Window  
 Support Inactivity Timeout  
 Support Run-Time Security Maintenance

Manager Override  
 Allow "Manager Override":   
 Override Duration:

Miscellaneous  
 Beep for "Access Denied"  
 Halt Code:

*ABC Template Chain*

Location of Library:

Support Logon Window  
 Support Run-Time Security Maintenance  
 Support Inactivity Timeout

Another setting that controls SuperSecurity is the Destination (EXE or DLL) in "Application Properties".

Manager Override  
 Allow "Manager Override":   
 Override Duration:

Miscellaneous  
 Beep for "Access Denied"

*Clarion/Legacy Template Chain*

**Support Logon Window** - Normally this setting should be On.

If you plan to determine the user number and name yourself, then you may want to turn it off. This is likely only if you are getting the user information with a network API library, or if you have your own logon window. After the user is determined via your own system, you must determine their `SSEC::User.No`, and save this value using `Security.Set_UserNo(value)`.

**Support Inactivity Timeout** - If you want the program to halt if the user hasn't done anything for a while, turn this on. This prevents the possibility of a high-level user walking away from their machine, and having someone else performing unauthorized tasks while they're gone. The options on the Inactivity tab specify the manner of the support.

**Support Run-time Security Additions** - If you turn this ON, the Run-time Security API calls will be inserted into all procedures in the APP. The options on the Run-time tab specify the manner of the support.

**Allow "Manager Override"** - If your user comes up against a security barrier, and a manager is usually available to override it, then you may want to turn this on (e.g.: doing a refund in a point-of-sale system). There are three possible settings:

**Never** - The manager cannot override when the user is denied access to a procedure or operation. This is the default.

**Always** - Whenever the user encounters an "Access Denied" message, they will be given the option of entering a manager override. (If `Security.CheckAccess` is called with a second parameter of `SSEC::Override:Never`, then "Always" will be temporarily overridden.)

**Use Local Setting** - This setting will leave the override option up to the individual security calls.

**Override Duration** - Once a manager has overridden a security barrier, the duration of the override could be one of the following:

**One-Time** - The override was a one shot deal. The original user is still active, and all security checks in the future will be applied only to the original user. If the user encounters this same point in the program again, he will need to have the manager override it again.

**Permanent** - The manager is now the active user on the system. It's as if the original user exited the program, then the manager restarted the program at logon instead.

**Beep for "Access Denied"** - If you want the system to BEEP when the access denied security window appears, then check this box. The default is "On".

**Halt Code** - This is the code that you wish to execute when the program is about to be HALT'ed. Just type the command here, as you would any other Clarion code.

**Type Tab**

Security Type:

With both types of security you can restrict user access at the procedure, browse operation, or control level.

**Security Type** - Controls whether you will be using Levels or Doors. There are buttons here leading to a brief explanation of Levels and Doors. For a more in depth description, see the What is Security section.

**Logon Tab**

Audit | Classes

General | Type | **Logon** | Inactivity | Run-time

Logon at program startup

Check for previous logon

User can change password during logon

Maximum Incorrect Tries:

Max. failures before lock:

Min. Password Length:

Initialize Super Tagging with User #

Backdoor Username:

Password Encryption Mask:

Mask UserName Condition:

Auto-Fill UserName from Previous Logon

Program Name:

ABC Template Chain

General	Type	Logon	Run-time	Audit	Inactivity
<input checked="" type="checkbox"/> Logon at program startup <input type="checkbox"/> Check for previous logon <input checked="" type="checkbox"/> User can change password during logon <input type="checkbox"/> Initialize Super Tagging with User # <input type="button" value="Auto-Logon with Network Username"/>					
Backdoor Username: <input type="text" value="BOXSOFT"/> Password Encryption Mask: <input type="text" value="!@#\$\$%^&amp;*"/> Maximum Incorrect Tries: <input type="text" value="3"/>					
<input type="checkbox"/> Auto-Fill UserName from Previous Logon Program Name: <input type="text"/>					
Code Before Init: <input type="text"/> Code Before Logon: <input type="text"/> Code After Valid Logon: <input type="text"/> Code After Failed Logon: <input type="text"/>					

*Clarion/Legacy Template Chain*

**Logon at program startup** - If you want the logon window to appear as soon as the program is started, then check this box. Otherwise, the logon window will be presented at the first security checkpoint. This is handy if there are very few secured areas in your program, so that you wouldn't normally have to logon.

**NOTE:** The Security.Logon procedure is called from the Init method of your EXE's first procedure (normally a Frame or Window). If your first procedure is actually a Source procedure, then you will have to call Security.Logon manually.

**Check for previous logon** - The security system enables your program to check for previous logon. That is, one application using the SuperSecurity templates calls another program using the RunSecurity Code template. The second program can check for previous logon so that the user need not logon multiple times.

If you check this box, then the Logon procedure will check to see if the program has been called by another SuperSecurity application before requesting a manual Logon.

**User can change password during logon** - This enables the password change support in the logon window. After they type in their name and password, they can hit [Change Password] instead of [OK]. They will be prompted to enter the new password twice. If they enter it properly, their password has been changed.

**Maximum Incorrect Tries** - This is the maximum number of times that a user can fail the logon attempt before the program halts. This is merely a nuisance feature, as they can just restart the program and try again. If you set this to zero then there is no maximum number of tries, and the user can continue trying indefinitely.

**Max. failures before lock** - If you want to lock users after a certain number unsuccessful logon

attempts, then specify the number (or expression containing that number) here. The failure count is reset with each successful logon.

**Min. Password length** - This ensures that the users don't enter new passwords that are too short. Specify the number or expression here.

**Password Expiration:**

**Support Password Expiration** - Check this box if you want your users' passwords to expire.

**Default for Maximum Password Age (Days)** - This is the default for any new users that are created. (It's assigned as the initial value via the dictionary settings and regular PrimeFields FileManager class method.) This value can be overridden for each user. If you're upgrading from an earlier version of SuperSecurity, then you should initialize the `PasswordMaxAge` field in the `SSEC::User` file during the data conversion.

**Check for Historical Uniqueness** - The password history is stored in `SSEC::PwdLog`. SuperSecurity can automatically check to ensure that the password is unique considering for a certain number of days, and/or a certain number of previous passwords. You can place any expression here, and a value of zero means "no check".

**Auto-Logon with Network Username** - With this turned ON, your program will attempt to access the current username via the Windows API. If it can find a valid username, then it will try to find a matching username in the `SEC::User` file (as if it were typed directly into the logon window). If it finds a match, the Logon window is skipped entirely, and it set the `User_` and `Username_` global variables accordingly.

**Initialize SuperTagging with User #** - Instructs the security system to generate a call to `UsrTag_` (`UserNo`). This should be turned on if you are using either SuperTagging or SuperQBE and you want the tags to be user specific. Also, the global extensions from at least one of these must be present.

**Backdoor Username** - It is suggested that you have a backdoor into your program so that you can support it if the security files become corrupted. The default is "BOXSOFT". If you blank this setting, then there will be no backdoor support included.

**Password Encryption Mask** - Many file drivers don't support encryption. To secure the passwords, we provided password encryption. We do a character XOR operation on the password to make it unreadable in the user file. If you want to make your program secure from other developers using the SuperSecurity templates, then you should change this from the default " !@#\$\$%^&\* ".

With the increased password size introduced in SuperSecurity 5.0, your mask can be longer. If it isn't long enough to cover the entire password field, it's automatically repeated using the `ALL` function.

This mask is also used by the "Check Previous Logon" function for encrypting data in the `WIN.INI` file.

**Mask Username Condition** - If you provide an expression here, then it will be checked for a non-FALSE value at run-time. In that case, it will add the `PASSWORD` attribute to the username entry control, just like the password entry control. This is useful, if you don't want anyone to see either usernames or passwords typed into the logon window.

**Auto-Fill Username from Previous Logon** - If you turn this ON, then it will remember the username

from one logon to the next, and the user will need to re-enter only their password. The username is stored in the WIN.INI file, as follows:

```
[SuperSecurity]
UserName:YourProgramName=YourUserName
```

For example, if your program is called "TEST" and your name is "Bob", then the entry would look like this:

```
UserName:TEST=Bob
```

This operation is actually performed within the SSEC::Logon procedure, which is part of your own program. Therefore, you can change the storage method if the current approach is unacceptable. Just search for `p_Security.AutoFillProgram`.

**This feature was added to the Clarion/Legacy chain in Super Security 6.05.**

**Code Before Logon** - This code is executed immediately before the Logon procedure. This is handy, because there is only one "Program Setup" embed. (This is not as important with C4+ABC.)

**Code After Valid Logon** - This is executed after a successful logon. This is handy, because there is only one "Program Setup" embed. (This is not as important with C4+ABC.)

**Code After Failed Logon** - This is executed after a failed logon. This is handy, because there is only one "Program Setup" embed. (This is not as important with C4+ABC.)

### **Inactivity Tab**

**Timeout Period (minutes)** - This is the maximum period of inactivity, after which the system automatically issues a HALT command.

**Logon After Inactivity Timeout** - If this is ON, then the inactive user will be presented with a Logon window, rather than being ejected completely out of the program. It saves the user the trouble of restarting the application.

### **Run-time Tab**

Run-time security allows your users (those with access to the proper door or level) to add security checkpoints to their program at run-time. This saves you from defining all of their security requirements during development. It also means that different sites can have different security.

The access points are associated with each procedure. For regular procedures (like Browsers), only the "General Access" setting is available. This will prevent unauthorized access to the procedure as a whole. For Form procedures (those with SaveButton template), Insert, Change, View and Delete can also be protected. For all of each of these access points, you can optionally support Manager Override.

The actual interface is provided by the SSEC::UpdateProcedure:Runtime form, which you *must* import into your APP from RUNTIMEL.TXA or RUNTIMED.TXA. If you don't like the look of the interface, you can change it to suit your needs.

There are two ways to invoke run-time security for window procedures. You can use a hot-key (like CtrlF10) or you can use a user event. You can arrange to generate this event yourself, or use the included control template to add a button to your toolbar or window.



**NOTE:** To invoke run-time security for reports, the user must hold down the Shift key while the report is being called.

If you want to update run-time access for one of your Source procedures, you can call *Security.UpdateRuntime* directly. For more information, see the API Reference.

**Invoked by** - This determines the manner in which the run-time security is invoked within Window procedures (e.g.: Browsers, Forms, etc.). You can choose "Hot Key" or "User Message". This setting applies globally to this APP. If you are creating a multi-APP project, then you must be sure to keep this setting consistent for all APPs.

**Hot Key** - This is the equate for the hot key that invokes the run-time security maintenance throughout the system. The default is CtrlF10.

**User Event** - This user event will invoke the run-time security maintenance throughout the system. Take care not to use an event that might be used by another template, or some of your own code. The default is EVENT:User+800. You can arrange to generate this event yourself, or you can use the Button for Run-time Security Maintenance in your toolbar.

**Administrator Level/Door** - This is the Door or Level required to maintain run-time security. If the user doesn't have access, the hot keys will be ignored, and the toolbar button will be hidden.

### Audit Tab

**Audit Backdoor User Activity** - Do you wish to audit the activity of the "Backdoor" user? This will normally be yourself (or other person associated with you), so it's probably unnecessary.

**Record Logon in Audit Log** - You may want to record when users log on to the system. These log entries will always show a program name of "Security" and procedure name of "Logon".

**Record Failed Attempts** - If you suspect that someone is hacking into your system by guessing passwords, then you could audit the failed logon attempts to see what username they're using, and what date and time they're making the attempts.

### Classes Tab

**Security Object** - By default, this will say "Security". If you need to override one or more of the Security method (usually the virtual ones), then change this to the name of your new object (e.g. MySecurity). Once you've changed the name here, there are two global embeds to which you must add code.

**Include INC file** - You can specify the name of your own security \*.INC file here, and it will be included in your global data at the proper location.

For the sake of this example, let's assume that you want to use variable filenames for the security files. You have created the variables, changed the dictionary settings to reflect this, and now you have to override the `PrepareFilenames` method.

**NEW in Super Security 6.04:** We added a new global embed, so that you can assign runtime filenames for the security files without going to the trouble of overriding the Security class. Look for "[SuperSecurity] Prepare Security Filenames".

### **Developing a Single EXE with no DLLs**

Add the following declaration code to the **"Global Data"** global embed:

```
MySecurity          CLASS(Security)
PrepareFileNames   PROCEDURE, DERIVED
END!CLASS
```

Then add the following code to the **"Program Procedures"** global embed:

```
MySecurity.PrepareFileNames PROCEDURE
CODE
MyUserFile = 'USER.DAT'
! etc.
```

### ***Developing a Multi-APP System***

Define your class using INC and CLW files. Assuming a root filename of "MYSEC" and a base files DLL of "MYFILES.APP", then MYSEC.INC would look something like this:

```
MySecurity          CLASS(Security),
MODULE('MYSEC.CLW'),
LINK('MYSEC.CLW', _ABCLinkMode_),
DLL(_ABCDllMode_)
PrepareFileNames   PROCEDURE, DERIVED
END!CLASS
```

MYSEC.CLW file would look something like this:

```
MEMBER('MYFILES.CLW') !This is your base APP
INCLUDE('MYSEC.INC'), ONCE
MySecurity.PrepareFileNames PROCEDURE
CODE
MyUserFile = 'USER.DAT'
! etc.
```

Finally, specify MYSEC.INC in the "Include INC file" global extension setting.

By the way, you can use this method for single APP systems too.

### ***After the Fact***

Anywhere that hand-code references the object called "Security", it must be changed to the name of your new security class object. For example, you'll get numerous compiler errors with UserEdit until you perform this task.

## 2.4 Procedure Security

### (Extension Template)

The ProcedureSecurity Extension template is used to protect a procedure. Its usage is very simple.

**Access Level/Door** - This should be a constant number or equate if you are using *Levels*, or a door equate from MYDOORS .CLW if you are using *Doors*.

**Allow Manager Override** - If you wish to allow manager override, check this box.

**Show "Access Denied" Message** - If you want the user to see an "Access Denied" message, then leave this box checked. Otherwise the user will get no indication that they have been denied (you may want this).

**Source of Message** - This can be "Local" or "Global". If you set it to "Local", then you will be able to enter the following fields. Otherwise, it will disable them.

**Message Text** - This is the local "Access Denied" message to be displayed. It is assumed that this is a string constant, unless it is prefixed by an exclamation point (!).

### Audit Access Tab

**Record Access in Audit Log** - If you wish to record this activity in your audit log, turn this ON.

**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It

can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

## 2.5 BrowseBox Update Security

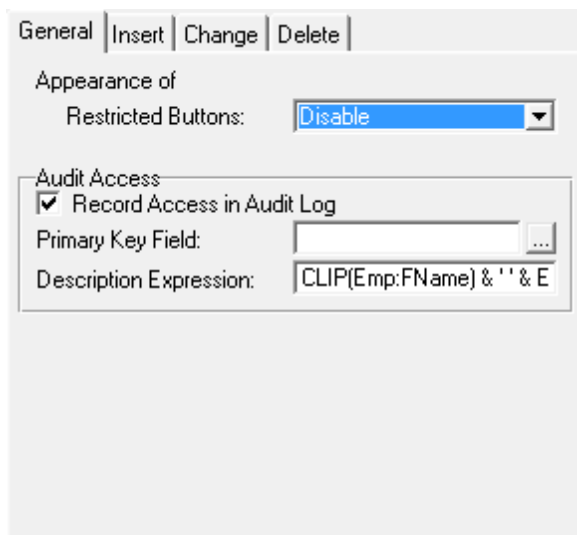
### *(Extension Template)*

The BrowseUpdateSecurity Extension template is implemented in much the same way as the ProcedureSecurity template, except that it has various additional options.

**NOTE:** If you can't find "BrowseUpdateSecurity" in the list of available extensions to add, then make sure that you have highlighted the "Browse on Filename" entry in your Extension list. For each BrowseBox on your window that you wish to protect with security, you must add another instance of the BrowseBox Update Security extension template.

The options are grouped into tabs, as follows:

### **General Tab**



**Appearance of Restricted buttons** -You have three options for the appearance of these buttons when they are unavailable:

**Hide** - The buttons for any restricted operations are hidden.

**Disable** - The buttons for any restricted operations are disabled.

**Show "Denied" Message** - The buttons are available, but the user will see an "Access Denied" message when they press them. If you wish to support manager override, you must set it to this option.

### **Audit Access**

**Record Access in Audit Log** - If you wish to record this activity in your audit log, turn this ON. Each of the Insert, Change and Delete tabs contains another checkbox to allow you do audit only certain requests.

**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

### Insert/Change/Delete Tabs

Except for the View options on the Change tab, these three tabs contain the same settings.

**Restrict Insert/Change/Delete** - Do you want to restrict this action? You may want to restrict only one or two of the operations.

**Audit Insert/Change/Delete** - Do you want to record this activity in the audit log?

**Allow "Manager Override"** - If your global manager override setting is "Use Local Setting", then the template will look to this setting. If the global setting is "Never" or "Always", then this setting will be ignored.

**Access Level/Door** - This is the required level or door for the user to perform the operation.

#### **"Insert/Change/Delete Denied" Message**

**Source of Message** - This can be "Local" or "Global". If you set it to "Local", then you will be able to modify the following settings. Otherwise, it will disable them.

**Message Text** - This is the local "Access Denied" message to be displayed. It is assumed that this is a string constant, unless it is prefixed with an exclamation point (!).

#### **View Instead (only on Change Tab)**

**View-only when Change is denied** - If you are restricting Change and the user is denied access, do you want them to be able to access the record in view-only mode? This will automatically set all entry controls to read-only and disable all buttons on the window.

**Access Level/Door** - If you want to support view, but they need special rights just to do that, you can specify the additional level/door here.



## 2.6 Form Update Security

### *(Extension Template)*

The *Form* update security and *BrowseBox* update security are much the same, except that *BrowseBox* prevents the form from ever being called, whereas *Form* aborts if the user doesn't have access to the requested action.

You must have a *SaveButton(Clarion)* control template on your Window to use this extension template.

The options are grouped into tabs, as follows:

### **General Tab**

#### **Audit Access:**

**Record Access in Audit Log** - If you wish to record this activity in your audit log, turn this ON. Each of the Insert, Change and Delete tabs contains another checkbox to allow you do audit only certain requests.

**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

### **Insert/Change/Delete Tabs**

Except for the View options on the Change tab, these three tabs contain the same settings.



General | Insert | Change | Delete

Restrict Change

Audit Change  
 Audit Field Edits Audit Field Settings

Access Level/Door:   
 Select Door from MyDoors.clw

Allow "Manager Override"

"Change Denied" Message  
 Source of Message:   
 Message Text:

View Instead  
 View-only when Change is denied  
 Access Level/Door (opt.):   
 Select Door from MyDoors.clw

Allow "Manager Override"

**Restrict Insert/Change/Delete** - Do you want to restrict this action? You may want to restrict only one or two of the operations.

**Audit Insert/Change/Delete** - Do you want to record this activity in the audit log?

**Access Level/Door** - This is the required level or door for the user to perform the operation.

**Allow "Manager Override"** - If your global manager override setting is "Use Local Setting", then the template will look to this setting. If the global setting is "Never" or "Always", then this setting will be ignored.

#### **"Insert/Change/Delete Denied" Message**

**Source of Message** - This can be "Local" or "Global". If you set it to "Local", then you will be able to modify the following settings. Otherwise, it will disable them.

**Message Text** - This is the local "Access Denied" message to be displayed. It is assumed that this is a string constant, unless it is prefixed with an exclamation point (!).

#### **View Instead (only on Change Tab)**

**View-only when Change is denied** - If you are restricting Change and the user is denied access, do you want them to be able to access the record in view-only mode? This will automatically set all entry controls to read-only and disable all buttons on the window.

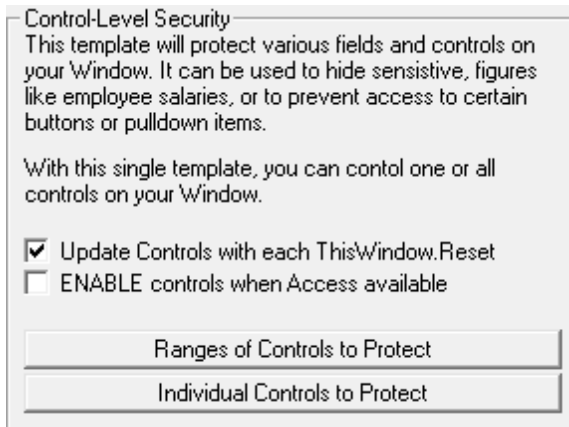
**Access Level/Door** - If you want to support view, but they need special rights just to do that, you can specify the additional level/door here.

**Allow "Manager Override"** - If your global manager override setting is "Use Local Setting", then the template will look to this setting. If the global setting is "Never" or "Always", then this setting will be ignored.

## 2.7 Protecting Controls on a Window

### *(Extension Template)*

The WindowControlSecurity Extension template is used to protect individual fields on a Window. You can protect individual controls, or consecutive ranges of controls.



If you wish to protect ranges of controls, press the [Ranges of Controls to Protect] button. Here you can add as many ranges as desired. In each case, only the first control is required. Specify the "Access Level/Door" and the "Appearance of Control" settings. Press the [OK] button when you're done.

Press the [Individual Controls to Protect] button to see a list of controls on your Window. For each of the desired controls, press the [Properties] button and specify the "Access Level/Door" and the "Appearance of Control" settings.

**Update Controls with each ThisWindow.Reset** - If there's a chance that the security status could change while you move around in the form, then you may want to have the control access status updated with each call to `ThisWindow.Reset`.

**ENABLE controls when Access available** - You may need the controls to be ENABLE'd explicitly when they are available. Turn this setting ON to force this attribute to be set.

## 2.8 Protecting Controls on a Report

### *(Extension Template)*

The ReportControlSecurity Extension template is used to protect individual fields on a Report. You can protect individual controls, or consecutive ranges of controls.

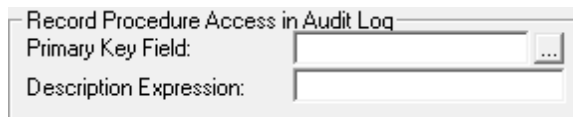
If you wish to protect ranges of controls, press the [Ranges of Controls to Protect] button. Here you can add as many ranges as desired. In each case, only the first control is required. Specify the "Access Level/Door" setting. Press the [OK] button when you're done.

Press the [Individual Controls to Protect] button to see a list of controls on your Report. For each of the desired controls, press the [Properties] button and specify the "Access Level/Door" setting.

## 2.9 Auditing Access to a Procedure

### *(Extension Template)*

Use this template if you are not using the Procedure Security template to restrict access to a procedure, but you still want to audit the access. After populating it into your procedure, you can also optionally enter any of the following settings:



Record Procedure Access in Audit Log

Primary Key Field:  ...

Description Expression:

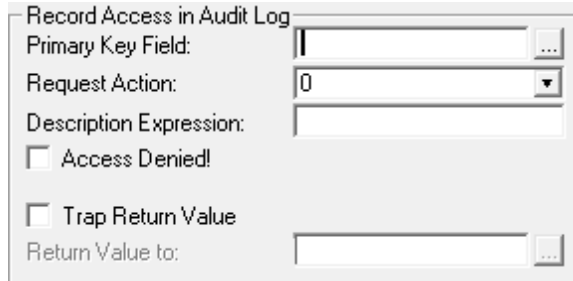
**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

## 2.10 Creating an Audit Trail Entry

### *(Code Template)*

Sometimes you will want to create an audit trail entry in one of your own embeds. This code template provides this facility. When you are inserting a new entry into an embed point, just choose AuditAccess from the SuperSecurity templates. Your other option is to call the Security.AddCall function yourself. For more information, see the API Reference.



The image shows a dialog box titled "Record Access in Audit Log". It contains the following fields and controls:

- Primary Key Field:** A text input field with a browse button (three dots) to its right.
- Request Action:** A dropdown menu currently showing the value "0".
- Description Expression:** A text input field.
- Access Denied!:** A checkbox that is currently unchecked.
- Trap Return Value:** A checkbox that is currently unchecked.
- Return Value to:** A text input field with a browse button (three dots) to its right.

**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Request Action** - Normally you will leave this as 0. If you want this audit point to represent InsertRecord, ChangeRecord, DeleteRecord or ViewRecord, then you can change it as appropriate.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

**Access Denied** - Do you wish for this log entry to be recorded as "Access Denied"?

**Trap Return Value** - Security.AddCall can return a pointer to the SSEC::Call record. This is handy if you want to update the call information later. If you need to trap this value, then turn this ON.

**Return Value to** - If you are trapping the return value from AddCall, then you must specify the destination variable for the value here.

---

## 2.11 Auditing Field Edits During a Change Operation

### *(Extension Template)*

*This template is available only in the Clarion/Legacy chain. Field-level auditing is incorporated into the Form Update Security template in ABC.*

It works with the *Form Update Security* template to keep a record of all fields that were modified during a "ChangeRecord" operation in a form. To populate it, you must highlight the existing "Restrict Access to Update Record in a Form" extension, then press Insert. You will find "ChangeAudit" within the SuperSecurity templates.

This template automatically audits all fields. If there are some that you don't need to track, then you can add them to the "Ignore" list.

## 2.12 Button for Run-time Security Maintenance

### *(Control Template)*

This control template will normally be populated onto your toolbar. It is used to invoke the Run-time Security maintenance mode for the currently active window. Of course, it will only work if the current user has sufficient access. In fact, the button will not be visible if the user doesn't have the necessary rights.

For this to work, you must have your global Run-time settings to generate a "User Event" rather than using a "Hot Key".

There are no prompts for this template.



## 2.13 Conditional Code

### (Code Template)

The ConditionalCode Code template checks the users security, and executes true and/or false code. The code can be a Procedure Call or Freeform Code. You can place this in any embed point.

The screenshot shows a dialog box titled "Security-controlled Code" with a tab labeled "Audit Access". The dialog contains several sections:

- Access Level/Door:** A text input field.
- Select Door from MyDoors.clw:** A dropdown menu.
- Allow "Manager Override":** An unchecked checkbox.
- "Access Denied" Message:**
  - Show "Access Denied" Message
  - Source of Message: Global (dropdown)
  - Message Text: Access Denied! (text input)
- Code to Execute:**
  - Type of Code: Call a Procedure (dropdown)
  - Granted Procedure: (dropdown)
  - Denied Procedure: (dropdown)
  - Granted Code: (text input)
  - Denied Code: (text input)

For an example of this, take a look at BrowseEmployee and PrintEmployees in TEST.APP in SUPER\EXAMPLES\SECUR\_XX. It is used to control the display of a salary field. A local variable is created to place on the list box and report. The code template decides whether to display the actual employee's salary or nothing.

## 2.14 Running a Program to "Check for Previous Logon"

### (Code Template)

For a program to be able to "Check For Previous Logon", it must be called with the correct parameters. To do this, use the "RunSecurity" Code template.

The most common way to do this would be to have a pulldown option on your application framework that would call the program. You must create an "Embed" point for the "Accepted" event. This will use the RunSecurity Code template.

All you need to specify is the program name. The security system does the rest. By the way, don't forget to enable "Check Previous Logon" in the program that you are calling.

### General Tab

The screenshot shows the 'General' tab of a dialog box. The text reads: "This code prepares the conditions so that a program called with the RUN command can 'Check for Previous Logon'." Below this, the 'Program Name' field contains the text 'UserEdit'.

**Program Name** - This is the name of the program that you wish to run (e.g.: PROGRAM.EXE).

### Restrict Tab

The screenshot shows the 'Restrict' tab of a dialog box. The 'Access Level/Door' field is set to 'eD\_Security'. Below it is a dropdown menu labeled 'Select Door from MyDoors.clw'. There is an unchecked checkbox for 'Allow "Manager Override"'. Under the 'Access Denied' Message section, the 'Show "Access Denied" Message' checkbox is checked, the 'Source of Message' dropdown is set to 'Global', and the 'Message Text' field contains 'Access Denied!'.

**Access Level/Door** - This should be a constant number or equate if you are using *Levels*, or a door equate from MYDOORS.CLW if you are using *Doors*.

**Allow Manager Override** - If you wish to allow manager override, check this box.

**Show "Access Denied" Message** - If you want the user to see an "Access Denied" message, then leave this box checked. Otherwise the user will get no indication that they have been denied (you may want this).

**Source of Message** - This can be "Local" or "Global". If you set it to "Local", then you will be able to enter the following fields. Otherwise, it will disable them.

**Message Text** - This is the local "Access Denied" message to be displayed. It is assumed that this is a string constant, unless it is prefixed by an exclamation point (!).

### **Audit Tab**



**Record Access in Audit Log** - If you wish to record this activity in your audit log, turn this ON.

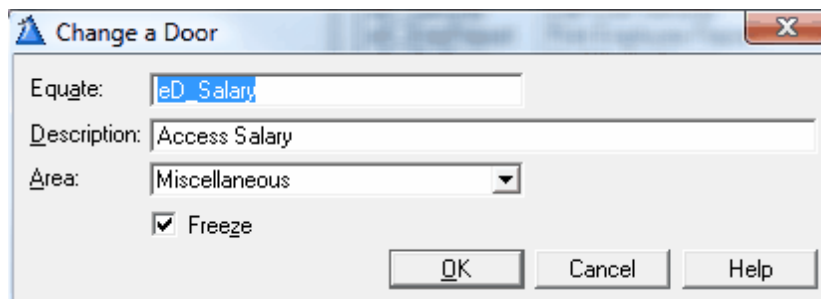
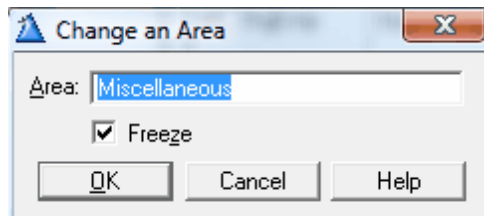
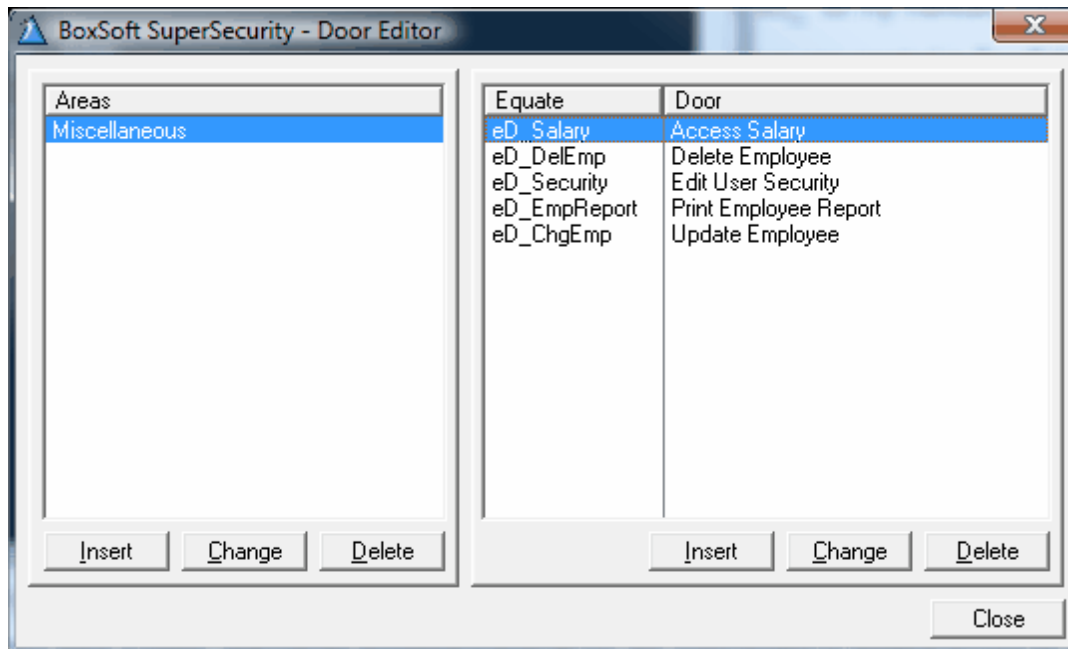
**Primary Key Field** - If this procedure is operating on a file that contains an identifiable primary key value, you can optionally specify the primary key field name here.

**Description Expression** - This is an optional description that will be stored with the audit log entry. It can be any valid Clarion expression, including everything from a single field name to a complex string concatenation.

## 3 Appendices

### 3.1 Support Programs - DoorEdit and UserEdit

#### DoorEdit



This program is used to maintain your Doors. If you are using Levels, then you can skip to the next section. You must create a DoorEdit that is catered to your dictionary. (If all of your dictionaries use the security file definitions, then you could use a single DoorEdit for all DCTs and their associated APPs.)

DoorEdit has the following responsibilities.

1. It maintains SSEC::Door and SSEC::DoorGroup. These files contains the door records used throughout the security system.
2. Upon exiting the program, you are given the option of generating MYDOORS.CLW. This INCLUDE file is designed to go into your application directory. It houses all of the door equates to be used throughout your application.

MYDOORS.CLW is created in the current directory. If you execute DoorEdit in a directory other than you application directory, then copy this file is in your application directory after its produced by DoorEdit. (You may also have to copy SSECUR\_.TPS so that it can be accessed by UserEdit and your application.)

I suggest that you use the prefix "eD\_" for your door equates, which is the default. (For consistency sake, I use "eL\_" for my manually created level equates.)

There is a field in both SSEC::Door and SSEC::DoorGroup called Freeze. It defaults to True if the door or door group was created in DoorEdit, and False if the entry was created using run-time security. You can tweak the Freeze value using DoorEdit, but your users cannot affect this value.

**NOTE:** Be very cautious about deleting doors. You cannot predict which doors may be referenced by the run-time security system at your many installed sites. It's better to create a door group called "Extinct", and to move the undesired doors into that group. When your users try to delete one of their own doors during run-time security maintenance, it verifies that it is unreferenced before allowing the deletion to proceed.

### ***Creating DoorEdit***

1. Create a new APP called DoorEdit in your application directory.
2. Tell it to use your dictionary.
3. Import \SUPER\LIBSRC\SECURITY\DOOREDIT.TXA into the new APP. When it complains about the dictionary error, ignore it and wait for the X to appear! Then hit [Close].
4. Delete empty modules, then renumber modules. (Both of these are under the Application menu.)

### **UserEdit**

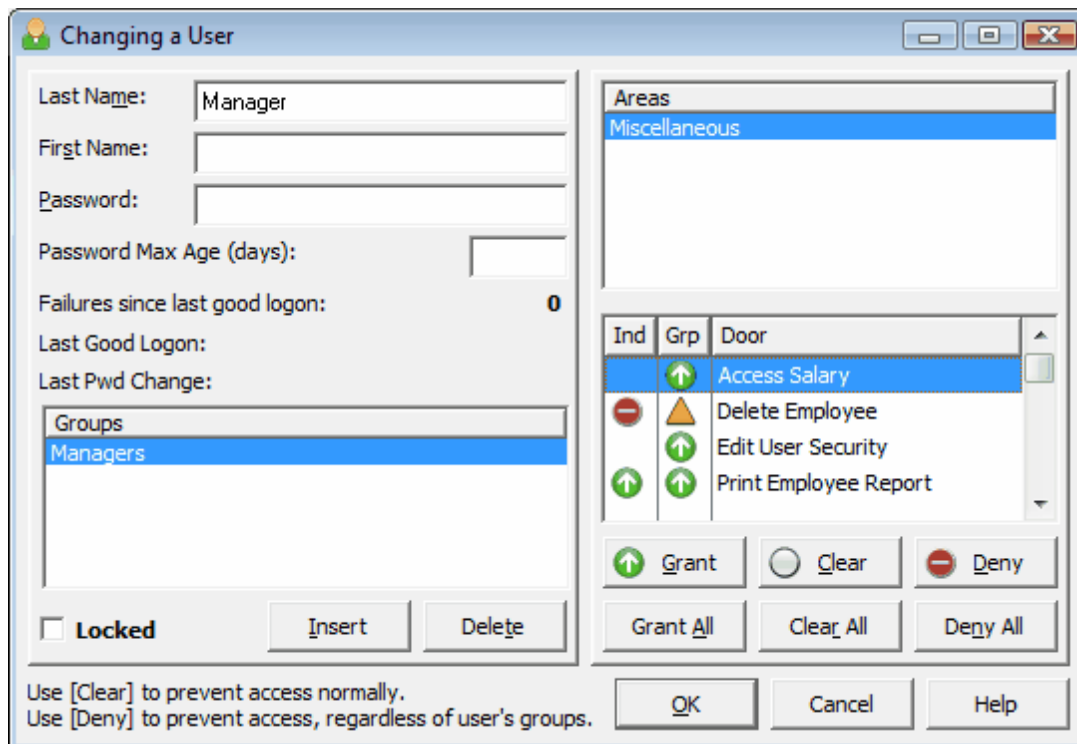
UserEdit is used by the "User Administrators" at your installed sites. It allows them to add new users and user groups, and to assign the users to any combination of user groups. Finally, they will assign access rights to the users and the groups.

You can think of user groups as simple users. They have a name, and can be assigned a level (if you are using levels) or granted access to any combination of doors (if you are using doors).

A user has a first and last name (only the last name is required), plus a password. Again, you can either assign a level or grant access to doors (depending on whether whether you are using levels or doors).

The most complex aspect of users is that of "inherited access". (This only applies if you are using doors.) If a user has not been granted access to a door as an individual, but they belong to at least one group that does have access, then the user gets access implicitly. You will see this when you are updating the user, because they don't have a checkmark in the "Ind" column, but they do have one in the "Grp" column.

If, for some reason, you don't want the individual user to have access, even though their group does, then you can "Deny" access to the individual. This displays an "X" in the "Ind" column for the door, while the checkmark in the "Grp" column changes from green to blue.



Note the above user's setup:

- The user's name is "Manager".
- They do not have a password.
- They are a member of the "Managers" user group.
- The "Managers" user group has been granted access to all of the doors (at least the ones visible on this window).
- The user has implicit access to the "Access Salary" and "Edit User Security" doors, because they are a member of the "Managers" user group.
- The user has access to the "Print Employee Report" door, both implicitly (through the "Managers" user group) and explicitly (personally granted here).
- The user has been explicitly denied access to the "Delete Employee" door, even though the

rest of the members of the "Managers" user group have access (unless explicitly denied as well).

### **Creating UserEdit**

1. Create a new APP called UserEdit in your application directory, and tell it to use your dictionary.
2. Import either SUPER\LIBSRC\SECURITY\USERED\_L.TXA or USERED\_D.TXA (depending on whether you are using Levels or Doors) into the new APP. When it complains about the dictionary error, ignore it and wait for the X to appear! Then hit [Close].
3. Delete empty modules, then renumber modules. (Both of these are under the Application menu.)
4. Go into the [Global] settings, then to [Extensions]. Now change these settings to match your desired setup. If you are calling UserEdit from another APP using SuperSecurity, then you may want to "Check for previous logon".
5. Go into the Global Embeds and change the "32000" in the Program Setup embed to match your security maintenance access level/group.
6. If you don't want the user administrator to see the users' passwords, you may want to add the "PASSWORD" attribute to the password field in SSEC::UpdateUser.

**NOTE:** When you ship your application, don't forget to include USEREDIT.EXE (or USEREDIT.DLL, if you've set it up that way).

DoorEdit is for the programmer's use only. There is no need to provide this to the end user. The only thing they need is the SSECUR\_.TPS file with all of the doors and door groups. It's strongly suggested that you include a few sample users along with the doors in SECUR\_.TPS. (If you are using levels, the sample file is not really required.)

## 3.2 Run-time Security Implementation

Run-time Security enables your users to add and maintain their own security checkpoints. This can save you effort during development time, and give you high marks from your users for providing a flexible system.

There are several steps required to implement run-time security:

1. The support procedure(s) for maintaining the run-time security must be imported into your application. (If you have a multi-APP system, it goes into your base support APP.) See Support Files and Procedures for more information.
2. Set the global settings to match your desired interface of Hot Key or User Event. Decide on a level/door for security maintenance. This will could be the same as the one chosen for user maintenance. See Security Global Extension for more information.
3. If you are using User Event, you can populate the Run-time Security Button control onto your toolbar. See Button for Run-time Security Maintenance for more information.



### 3.3 Multi-APP Development using LIBs/DLLs

If you are using the Security 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\SECUR\_L2 and SECUR\_D2. The TEST.APP in each of these directories shares the basic security support in UserEdit, along with calling the SSEC::UserEdit procedure directly.

#### **LIB/DLL with the Security Code (Base APP)**

1. Create or open the APP destined to be a DLL/LIB.
2. Choose "Application / Properties ..." from the pulldown menu.
3. Ensure that your "Destination Type" is set to "Dynamic Link Library (.DLL)" or "Library (.LIB)", then press [OK].
4. Press the [Global] button, then the [Extensions] button.
5. If you have already included the SecurityGlobal extension template, then skip this step. Otherwise, press the [Insert] button, and Select "SecurityGlobal" under "Class SuperSecurity".
6. Ensure that the "Location of Library" is set to "Internal".
7. Review the rest of the settings for your preferences, then press [OK] to exit the "Extensions and Control Templates" window.
8. Go to the "File Control Flags" tab, and turn ON "Generate all file declarations". If you are creating a DLL, turn ON "Export all file declarations". Then press [OK] to exit the "Global Properties" window.
9. Make the DLL/LIB. (Press the [Lightning] on the button bar, select "Project / Make" from the pulldown, or press Ctrl-M.) The LIB file will be found in the first directory specified in the \*.LIB entry of your redirection file. 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 Security Code (Dependent APP)**

1. Create or open the APP that will call the procedures in the APP described above. Press the [Global] button, then the [Extensions] button.
2. If you have already included the SecurityGlobal extension template, then skip this step. Otherwise, press the [Insert] button, and Select "SecurityGlobal" under "Class SuperSecurity".
3. In the Security extension, ensure that the "Location of Library" is set to "External", and verify that the rest of the available settings match those of your base APP. Then press [OK] to exit the "Extensions and Control Templates" window.

4. Turn on the "Generate global data as EXTERNAL" check box.
5. Go to the "File Control Flags" tab. Within the "File Attributes" box, set the "External" setting to "All External". Then turn ON "All file are declared in another APP". Press [OK] to exit the "Global Properties" window.
6. Go to "Application / Insert Module" on the pulldown menu. Select "External DLL" or "External LIB". This must match your decision when you created the DLL or LIB.

Change the name to match your module (e.g.: UserEdit.LIB). (It's always a LIB, even if you are referring to a DLL. This is because a DLL still produces a LIB for linking.) Press [OK] to save the Module.

Press the "Module" tab to see the procedures grouped by module. Highlight your newly created module.

7. If you want to call one or more procedures from your newly declared external module, press the "Module" tab to see the procedures grouped by module. Highlight your newly created module.

Choose "Procedure / New" from the pulldown menu, or press the [Insert] key. If you are doing this from the "Procedure" tab, then you will have to specify that it uses the "External" procedure type, and that it is contained in your new external module.

Type the name of the desired procedure. (If you are using UserEdit, you could call SSEC::UserEdit.) If your procedure has a prototype (most do not) enter it here, then press [OK] to save it.

8. Make the APP. (Press the [Lightning] on the button bar, select "Project / Make" from the pulldown, or press [Ctrl-M].)

Remember, if you are using DLLs, then all references must be down. For example, you have three APPs: Main, Misc and Base. Misc and Base are DLLs, while Main is an EXE. Main calls routines in Misc and Base. Misc calls routines in Base. Therefore, Base must be made first, then Misc, then Main. If Misc did not call anything in Base, then you could make Misc before Base, 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 looks at all modules (EXE, OBJs and LIBs) 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 when it is made.

### 3.4 SQL and Super Security

If your database is being stored on a SQL server, then you may want to move the Security data files there as well.

**NOTE:** TPS files are somewhat secure unto themselves. Once you place these files on your SQL server, you'll have to take additional efforts to restrict their use!

One possible approach is to have them accessible only by the DBA and one other "secure" user. All others (including the "public") cannot access them. Then define the security files to use this secure user when accessing the database from within your program. Anyone other than the secure user or dba who attempts to access the files in other systems (e.g. via separate SQL query tools, ODBC, etc.) will be denied.

If you want an example of how this is setup, you can take a look at the example in EX\_ABC\SECUR\_DS. Take note of the file definitions in the dictionary, CONVERT.PRJ, CONV\*.CLW, and SQLLOGON.INC.

Here is one possible approach to convert your files from their current incarnations into SQL versions. (It's not the only solution, but it will do the job.)

1. Copy and Paste existing Security files to File\_OLD, with prefix Pre\_O.
2. Change fields in pre-existing files (not the copies from #1):
  - a. STRING(n) to CSTRING(n+1)
  - b. MEMO(n) to CSTRING(n+1)
  - c. Depending on your back-end, you may want to assign an "External Name" (on the Attributes tab) for some of the fields. For example, "Date" may be a reserved word.
3. Change pre-existing files (not the copies from #1):
  - a. Owner Names to match your system. Although a !Variable is best, you can also use an !EQUATE or literal string (e.g. Server,Database,User>Password).
  - b. Full Pathnames to dbo.SSECUR\_File, or any other name that suits your fancy.
  - c. Turn OFF the Reclaim and Encrypt attributes.
  - d. Change the driver to match your SQL variant (e.g. "Microsoft SQL").
4. Create conversion programs for each file. One way is to Browse the "File\_OLD" from the dictionary, and then tell it you want to create a conversion program. See EX\_ABC\SECUR\_DS for an example which calls a bunch of these from one program. Also watch for these issues:
  - a. CLIP(STRINGs) and CLIP(MEMOs).
  - b. Use ADD instead of APPEND.
  - c. Don't STREAM or FLUSH.
  - d. Don't BUILD.
5. Once you've ensured that the conversion worked correctly, you can remove the "File\_OLD" copies from your dictionary.

## 3.5 Disabling Security

There may be situations where you want to prevent the security system from kicking in. This is a two step process:

1. Tell the global template not to logon at start-up. This prevents the logon window from appearing until the user reaches a protect area.
2. Set the user number to -1 (a.k.a. the backdoor) with the command **Security.Set\_UserNo(-1)**. Make sure you do this before a protected area is encountered (at program startup is best, of course).

## 3.6 Interface Modification and Translation

We've moved all displayable strings to **StAbSec.trn**. This file contains equates for messages, titles, buttons and prompts throughout the Super Security 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. Or you can have your own version of the file somewhere in the RED path before LIBSRC.

**NOTE:** If you have copied **StAbSec.trn** into your APP directory, then upgraded to a new version of Super Security, don't forget to check the new TRN file for changes. If you don't keep your own copy of this file up to date, then you will get compiler errors.

**StAbSec.trn** used to have Window structures as well, but as of version 5.0 these have all be converted into regular Window procedures that you import into your APP from TXAs. You can change the text for these windows directly in your APP.

If you want your program to support multiple languages at run-time, then you have to override the Security.Translate function. This function takes one parameter, the name of the equate in **StAbSec.trn** (e.g. '**SSEC:Message:Title**'), and it returns the corresponding text. Without overriding it, the function just returns the value of the equate, which is sufficient to support a single language.

To support multiple languages, you must override the Security class (see the "Classes" tab in the Global Extension Template), and then write your own Translate function. Take a look at **StAbSec.c1w** to see how the existing function is written. Just check for the standard equate names, and return an appropriate string value for the current language.

## 3.7 Example Programs

There are four example programs provided with these templates. They are essentially the same, except that two use levels while the other two use doors. You can find them in:

```
SECUR_L1 - Levels with USEREDIT.EXE
SECUR_L2 - Levels with USEREDIT.DLL
SECUR_D1 - Doors with USEREDIT.EXE
SECUR_D2 - Doors with USEREDIT.DLL
SECUR_DS - Doors with USEREDIT.EXE (SQL)
```

The Doors examples use MYDOORS.CLW, while the Levels example use constant numbers.

The SSECUR\_.TPS file in each directory contains the following users:

```
Clerk      : L=1      : D=
Manager    : L=4      : D=1,2,3,4,5
```

**NEW in Super Security 6.05:** SECUR\_DS contains a Microsoft SQL Server example. There is a "source" SSECUR\_.TPS, but you must run a conversion to transfer the records to your SQL database server. It's assumed that you're using the ubiquitous Northwind example database, and our example program references the Employees file within it. Edit SQLLOGON.INC to specify your server, database (probably "Northwind" and logon information, then make and run CONVERT.PRJ.

### Global

- The Location of Library is set to "Internal" for SECUR\_x1, and "External" for SECUR\_x2.
- The Security Type set to "Levels" for SECUR\_Lx, and "Doors" for SECUR\_Dx.
- Prior to SuperSecurity 5.0, a custom STABSEC.TRN file was used to tweak the logon window. The new template installation program has removed this file, and the window modifications are directly within the SSEC::Logon procedure.
- The run-time security is enabled and told utilize a User Event.
- The inactivity time-out is set to 10 minutes.

### Main

- Window: "View Audit Log" has been added to the pulldown.
- Window: The Run-time security button has been added to the toolbar.
- Window and Embed: The "SecurityRun" code template is used to call UserEdit edit in SECUR\_x1. The SSEC::UserEdit procedure is called directly in SECUR\_x2.
- Window and Embed: The *Security.Logon* procedure is called from a pulldown item to allow the user to logon under a different name without exiting the program. You *must* use a Source embed for this, rather than a procedure call.

### **BrowseEmployee**

- Extension: Restrict Access to Update Records from a Browse. Only the delete operation is protected.
- Embed: Format an element of the Browse Queue. The "ConditionalCode" code template is used to display the salary only if the user has valid access. We use a local variable to place in the BrowseBox. (See Loc:Salary in the local Data.)

### **UpdateEmployee**

- Extension: Restrict Access to Update Record in a Form. Change is restricted, with View-only supported. Also, Change and Delete operations are audited.
- Extension: ChangeAudit template is used to provide field-level audit trail during ChangeRecord operations.
- Extension: Restrict Access to Window Controls. Salary prompt and field are hidden using this technique.

### **PrintEmployees**

- Extension: Restrict Access to the Procedure. Level 2 (or eD\_PrintEmp) required to access the report.
- Embed: Before Printing Detail Section. "ConditionalCode" code template is used to display the salary only if the user has valid access. We use a local variable to place on the Report. (See Loc:Salary in the local Data.)

## 3.8 API Reference

Here are some of the important properties and methods of the Security class. For a complete list, please see the source code in STABSEC\*.INC and STABSEC\*.CLW.

### Class Properties

#### **Security.FullAccess**

If you want your users to logon, but you temporarily want to allow unlimited access throughout the program, then set this to True. It's value is checked with each call to Security.CheckAccess, so you can change the value on-the-fly.

### Class Methods

```
Security.Set_UserNo ( LONG UserNo, BYTE Sync=1 )
Security.Get_UserNo ( BYTE Sync=1 ), LONG
```

This is the current user number. If the user has not yet logged on, the value will be zero. If you use the backdoor username to logon, the value will be -1.

The Sync parameter is optional. If you've already called Security.Sync.Wait, then you can pass False here to indicate that the synchronization is already in effect.

```
Security.Set_UserName ( STRING UserName, BYTE Sync=1 )
Security.Get_UserName ( BYTE Sync=1 ), STRING
```

This is the current user name in this format:

```
LEFT(CLIP(U_:FName) & ' ' & U_:LName)
```

If the user has not yet logged on, it will be blank. If you use the backdoor to logon, the value will be '[Backdoor]' (unless you have changed it in STABSEC.TRN).

The Sync parameter is optional. If you've already called Security.Sync.Wait, then you can pass False here to indicate that the synchronization is already in effect.

```
Security.AddCall ( STRING Program, STRING Procedure, <LONG Request>, <STRING File>,
<LONG PrimaryKey>, <STRING Description>, <BYTE AccessDenied>), LONG, PROC
```

This procedure is responsible for adding audit log entries to the SSEC::Call file. It automatically determines the user, date and time. If you trap the return value, you can use it in a future call to Security.PutCall.

**Program** - The name of the program (without extension)

**Procedure** - The procedure name

**Request** (*optional*) - The requested action (e.g.: InsertRecord, ViewRecord, or zero)

**File** (*optional*) - The label of the data file, passed as a string.

**PrimaryKey** (*optional*) - The primary key field value (e.g.: Cus:No)



**Description** (*optional*) - A description (up to 100 characters)

**AccessDenied** (*optional*) - Specifies whether this activity was denied because of insufficient access.

Examples:

```
Call# = Security.AddCall('Test', 'UpdateEmployee', ChangeRecord, 'Employee', Emp:No, CLIP
(Emp:LastName) & ', ' & Emp:FirstName, 1)
Security.AddCall('UserEdit', 'SSEC::Main')
```

**Security.CheckAccess** ( **SHORT Door**, <BYTE Override>, <STRING AccDenMsg>, <BYTE UseGlobMsg>), BYTE

This function is responsible for checking for valid access. It returns `True` if the user has access, and `False` if he doesn't.

*Level/Door* - This is the access being checked. It is the only required parameter.

*Override* (*optional*) - This optional parameter indicates whether an override is available. Its meaning is different, depending on the global "Manager Override" setting:

If it is set to "Never", then this parameter is ignored.

If it is set to "Always", then passing this parameter as `False` (or `SSEC::Override:Never`) will prevent the override option. If you omit the parameter or pass `True`, the override option is available.

If it is set to "Use Local Setting", then this parameter is the controlling force.

*AccDenMsg* (*optional*) - This is the local access denied message. If it is included, the global message will not be displayed. This parameter is optional.

*UseGlobMsg* (*optional*) - If the "Access Denied" parameter is omitted, then this indicates the current operation being performed: `InsertRecord`, `ChangeRecord`, `DeleteRecord`, or zero (general access). This parameter is used to display the appropriate global message. If all message parameters are omitted, then it is assumed that a zero was passed for this parameter.

To check for access with the override default:

```
IF Security.CheckAccess(Door) THEN ...
```

To "silently" determine if the user has access:

```
IF Security.CheckAccess(Door, 0) THEN ...
```

To supply an alternative "Access Denied" message:

```
IF Security.CheckAccess(Door,, 'Not Allowed!') THEN ...
```

To manually check for insert access with the global message defaults, use:

```
IF Security.CheckAccess(Door,,, InsertRecord) THEN ...
```

There are two global embeds available within this method: "CheckAccess is beginning" and "CheckAccess is ending". If you put your code in the "Ending" embed, then you know which door is

accessed via `p_Door`, whether the user has access to the door via `L::RetVal=True/False`, and the current user number via `SELF.GetUserNo()`. Note that neither the User nor Door files are open when this method is called.

#### **Security.CheckDoorUsed ( ), BYTE**

This function is used to check for SSEC::Procedure records referencing the current SSEC::Door record in memory. If at least one exists, a warning message is displayed and the function returns True. Otherwise, the function returns False.

Examples:

```
IF Security.CheckDoorUsed() THEN CYCLE.

IF NOT Security.CheckDoorUsed() THEN DELETE(SSEC::Door).
```

#### **Security.CheckRuntimeAccess ( STRING Program, STRING Procedure, <LONG Request>, <BYTE NoMessage>, <\*BYTE Viewing> ), BYTE**

This procedure is similar to CheckAccess, except that it uses the run-time security settings instead of the pre-programmed settings. If this procedure is being called from a form and the user is restricted from changing records, then this procedure may change the value of SSEC::ViewRecord. It returns True if the user is allowed to continue.

*Program* - The name of the program (without extension)

*Procedure* - The procedure name

*Request* (optional) - The requested action (InsertRecord, ChangeRecord, DeleteRecord, or zero)

*NoMessage* (optional) - Suppress any "Access Denied" message that might occur.

*Viewing* (optional) - When being called from an update form with a local "Viewing" status variable, set this variable TRUE if view-only mode applies. (It won't set it to False.)

Examples:

```
Security.CheckRuntimeAccess('Test', 'UpdateEmployee', GlobalRequest)

Security.CheckRuntimeAccess('Test', 'BrowseEmployee', 0)
```

#### **Security.CheckUsersExist ( <BYTE GroupFlag> ), BYTE**

This function returns True if any users exist. It is used by UserEdit to determine whether to disable the Insert button on the SSEC::UpdateUserGroup form.

*GroupFlag* (optional) - This indicate whether it should really be looking for User Groups instead of Users.

Example:

```
IF ~Security.CheckUsersExist() THEN DISABLE(?Insert).
```

#### **Security.CheckUserGroupsExist ( ), BYTE**

This function returns True if any user groups exist. It is used by UserEdit to determine whether to disable the Insert button on the SSEC::UpdateUser form.

Example:

```
IF ~Security.CheckUserGroupsExist() THEN DISABLE(?Insert).
```

```
Security.CompareEdit ( LONG CallNo, STRING File, STRING Field, *? OldValue, *? NewValue )
```

This procedure supports the field-level auditing during ChangeRecord operations. It compares the value of the two variables, and creates an audit log entry in SSEC::Edit if the values do not match. The newly created record is related to the SSEC::Call record created during the recent execution of Security.AddCall.

*CallNo* - This is the call number returned from the earlier execution of the Security.AddCall() function.

*File* - This is the label of the data file.

*Field* - This is the label of the field.

*OldValue* - This is the original value of the field, saved on the way into the form.

*NewValue* - This is the value current value of the field.

Example:

```
Security.CompareEdit(Loc:CallNo, 'Employee', 'Name', Loc:EmpName,
Emp:Name)
```

```
Security.EncryptPassword ( STRING ), STRING
```

This function uses the password encryption mask that you entered in the global extension template to encrypt and decrypt the password. Simply pass it a string, and it will return it en/decrypted. The string must be no longer than 8 characters, and it will always return an 8 character string.

Example:

```
Loc>Password = Security.EncryptPassword(Glo>Password)
```

```
Security.GetCallRequest ( ), STRING
```

This function returns the name of the action described by the current value of the SCall\_:Request field. It is used for displaying the request in SSEC::BrowseAudit. You can change these descriptions in STABSEC.TRN.

```
Security.GetCallUsername ( ), STRING
```

This function returns the name of the user specified by the current value of the SCall\_:UserNo field. It is used for displaying the username in SSEC::BrowseAudit.

```
Security.GetFieldName ( LONG FieldNo ), STRING
```

This function returns the name of the field associated with the FieldNo parameter. It fetches the corresponding record from SSEC::Field. It is used for displaying the file name in SSEC::BrowseAudit.

*FieldNo* - This is the value of SField\_:No, or a corresponding foreign key field from another file.

Example:

```
Loc:Field = Security.GetFieldName(Loc:FieldNo)
```

**Security.GetFileName ( LONG FileNo ), STRING**

This function returns the name of the file associated with the FileNo parameter. It fetches the corresponding record from SSEC::File. It is used for displaying the field name in SSEC::BrowseAudit.

*FileNo* - This is the value of SFile\_:No, or a corresponding foreign key field from another file.

Example:

```
Loc:File = Security.GetFileName(Loc:FileNo)
```

**Security.GetNetUsername ( ), STRING**

This function returns the network username (if available to the Windows API).

Example:

```
Loc:Username = Security.GetNetUsername()
```

**Security.GetProcName ( LONG ProcNo ), STRING**

This function returns the name of the procedure associated with the ProcNo parameter. It fetches the corresponding record from SSEC::Procedure. It is used for displaying the procedure name in SSEC::BrowseAudit.

*ProcNo* - This is the value of SProc\_:No, or a corresponding foreign key field from another file.

Example:

```
Loc:Procedure = Security.GetProcName(Loc:ProcNo)
```

**Security.GetProgName ( LONG ProgNo ), STRING**

This function returns the name of the program associated with the ProgNo parameter. It fetches the corresponding record from SSEC::Program. It is used for displaying the program name in SSEC::BrowseAudit.

*ProgNo* - This is the value of SProg\_:No, or a corresponding foreign key field from another file.

Example:

```
Loc:Program = Security.GetProgName(Loc:ProgNo)
```

**Security.Init**

This procedure initializes the Security class. It must be called when the program first begins.

Example:

```
CODE
Security.Init
IF NOT Security.Logon THEN RETURN.
MainMenu
Security.Kill
```

**Security.Kill**

This procedure cleans-up the Security class. It must be called when the program is finishing.

Example:

```
CODE
Security.Init
IF NOT Security.Logon THEN RETURN.
MainMenu
Security.Kill
```

#### **Security.Logon ( <LONG Override> ), LONG, PROC**

This is the main vehicle for logging-on to the program. In addition to providing the user interface for the logon window, it also checks for previous logons (depending on your global extension settings). If the user logged-on successfully, the return value is the user number. Otherwise, it returns zero.

You can call this procedure yourself to allow a new user to logon. See the example program's "File/Logon" pulldown item for a sample of this usage.

*Override* (optional) - The optional parameter is only used when Logon is called by CheckAccess for a manager override verification.

Examples:

```
Security.Logon

IF NOT Security.Logon() THEN HALT.
```

#### **Security.PostRuntimeEvent**

This procedure is called by the "Invoke Run-time Security" control template. It posts the specified run-time user event to the currently active window. It takes no parameters and returns nothing, as all necessary settings are already stored in the Security class by the Init procedure.

Example:

```
Security.PostRuntimeEvent
```

#### **Security.PrepareActivityFrame**

This procedure is used to support the inactivity time-out feature. It ensures that the frame procedure has a timer event for checking inactivity. It is called after the specified frame procedure opens its window. It takes no parameters and returns nothing, as all necessary settings are already stored in the Security class by the Init procedure.)

Example:

```
Security.PrepareActivityFrame
```

#### **Security.PrepareFileNames**

The method is called from Security.LoadQs, which is executed before Security.Logon in your Main procedure. It's an empty virtual procedure that you can override to set your own variable filenames for the security files.

**NEW in Super Security 6.04:** We added a new global embed, so that you can assign runtime filenames for the security files without going to the trouble of overriding the Security class. Look for "[SuperSecurity] Prepare Security Filenames".

*Alternatively*, you can define a class in a global embed like this:

```
MySecurity          CLASS (Security)
PrepareFileNames  PROCEDURE, DERIVED
                  END!CLASS
```

Then add the method to your "Program Procedures" global embed. It will look something like this:

```
MySecurity.PrepareFileNames PROCEDURE
  CODE
  !Assign filenames here
```

Finally, tell the security global extension that the security object is called "MySecurity".

#### **Security.PreparePreviousLogon**

This procedure is used by the "Check Previous Logon" support. It should be used immediately before calling an EXE with the RUN command. Of course, the called EXE must be told to check for previous logon.

Example:

```
Security.PreparePreviousLogon
RUN( 'USEREDIT' )
```

#### **Security.PrepareRuntimeButton**

This procedure is used by the Run-time Security support. It is called after opening the window containing the run-time maintenance invoking button. If the user doesn't have the right to perform run-time security maintenance, then the control will be hidden.

If you call the Security.Logon procedure to allow a different user to logon, you may want to call this Security.PrepareRuntimeButton immediately after, to ensure that it reflects the current user's rights.

Example:

```
Security.PrepareRuntimeButton(?InvokeRuntimeSecButton)
```

#### **Security.PrepareRuntimeWindow**

This procedure is used by the Run-time Security support. It is called after opening the window in all procedures in the APP (except for the main frame). If you've specified to use a Hot Key to invoke run-time security maintenance, this attaches the key as an AlertKey to the window.

Example:

```
Security.PrepareRuntimeWindow
```

#### **Security.PrepareViewWindow ( \*BYTE Viewing, <LONG CancelButton> )**

This procedure is used to prepare an update form during a ViewRecord operation. It is called after opening window in all Update forms in the APP (i.e.: any windows containing the SaveButton control template). If SSEC::ViewRecord is True, then the procedure will set all entry controls (e.g.: ENTRY, MEMO, COMBO, SPIN) to read-only, and disable all buttons.

*Viewing* - The passed BYTE variable is set to True if it's performing a ViewRecord operation, The local variable is then used to remember whether the procedure is in "View" mode.

*CancelButton* (optional) - This is the Cancel button control. This button will not be disabled.

Examples:

```
Security.PrepareViewWindow(Loc:Viewing)
Security.PrepareViewWindow(Loc:Viewing, ?Cancel)
```

#### **Security.Purge ( LONG Date, <BYTE Stream> )**

This procedure purges old entries from the audit log (SSEC::Call and SSEC::Edit).

*Date* - This is the date before which you want to delete all entries.

*Stream* (optional) - Do you want to stream the delete operation. The files will be locked during this period, so busy multi-user systems may not want to use this.

Example:

```
Security.Purge(TODAY()-60, True)
```

#### **Security.PutCall ( LONG CallNo, LONG Response, <LONG PrimaryKey>, <STRING Description> )**

This procedure is used in conjunction with Security.AddCall. If the user cancels the operation, changes the primary key field, or changes the description, this gives the system a change to update the audit trail with the latest information.

*CallNo* - This is the call number returned from the earlier execution of the Security.AddCall() function.

*Response* - This is either RequestCancelled or RequestCompleted. You can usually use LocalResponse or GlobalResponse.

*PrimaryKey* (optional) - The primary key field value (e.g.: Cus:No)

*Description* (optional) - A description (up to 100 characters)

Example:

```
Security.PutCall(Loc:CallNo, LocalResponse, Emp:No, CLIP(Emp:
  LastName) & ', ' & Emp:FirstName)
```

#### **Security.ResetOptions**

This procedure is generated into APP\$SEC.CLW (where "APP" is the first four characters of your APP name). It sets the options within the Security object using the settings specified in the global extension.

Usually you will not have to call this yourself. However, if you have specified variable or field names for some of the Security settings, and if the values of those variables/fields have changed, then you can call this method to reset the values in the Security object to the current values of the variables.

Example:

```
Security.ResetOptions
```

#### **Security.RestoreUser ( <BYTE GroupFlag> )**

**Security.RestoreUserGroup**

This procedure restores the Access and UserInGroup records for the current user or user group in memory, based upon the state when SaveUser was called. It is used by UserEdit to handle the Cancel button on the SSEC::UpdateUser and SSEC::UpdateUserGroup forms.

*GroupFlag* (optional) - This indicates whether it should really be looking for User Groups instead of Users. RestoreUserGroup just calls RestoreUser(True).

Example:

```
IF (OriginalRequest = InsertRecord OR OriginalRequest = ChangeRecord)
    AND LocalResponse = RequestCancelled
    Security.RestoreUser
END!IF
```

**Security.SaveUser ( <BYTE GroupFlag> )****Security.SaveUserGroup**

This procedure restores the Access and UserInGroup records for the current user or user group in memory, based upon the state when SaveUser was called. It is used by UserEdit to handle the Cancel button on the SSEC::UpdateUser and SSEC::UpdateUserGroup forms.

*GroupFlag* (optional) - This indicates whether it should really be looking for User Groups instead of Users. SaveUserGroup just calls SaveUser(True).

Example:

```
IF LocalRequest = InsertRecord OR LocalRequest = ChangeRecord
    Security.SaveUser
END!IF
```

**Security.TakeActivityEvent**

This procedure is used to track user activity to support the Inactivity Time-out. It watches for events that indicate that the user is "active" on the machine. It updates internal variables where applicable. The procedure is normally called at the top of all the ACCEPT processing loops in windows, and the "NextRecord" loops in Processes and Reports.

Examples:

```
Security.TakeActivityEvent
```

**Security.TakeActivityFrameEvent**

This procedure is used to support the Inactivity Time-out. It watches for the timer event in the main frame. When the event occurs, the procedure checks for recent user activity. If they have been inactive for too long, the program will be halted.

Example:

```
Security.TakeActivityFrameEvent
```

**Security.TakeRuntimeEvent ( STRING Program, STRING Procedure, BYTE UpdateForm )**

This procedure is used to recognize when the run-time security maintenance is being invoked (by either the Hot Key or User Event). If appropriate, it calls SSEC::UpdateProcedure:Runtime.



*Program* - The name of the program (without extension)

*Procedure* - The procedure name

*UpdateForm* - This specifies whether the procedure is an update form (i.e.: a Window with the SaveButton control template). If so, then the user will be allowed to update all the run-time security doors for that procedure, not just the general access door.

Examples:

```
Security.TakeRuntimeEvent('Test', 'UpdateEmployee', 1)
```

```
Security.TakeRuntimeEvent('Test', 'BrowseEmployee', 0)
```

**Security.TakeViewWindowEvent ( BYTE Viewing, <LONG CancelButton> ), BYTE**

This function is used to ensure that no "unacceptable" events are processed when the user is in an update form in ViewRecord mode. It's called at the top of the ACCEPT processing loop. It returns True if the event should be ignored.

*Viewing* - This parameter tells the function whether the window is currently in ViewRecord mode. If this is zero, then the function will always return false.

*CancelButton* (optional) - This is the Cancel button control. EVENT:Accepted for this control will not be ignored.

Example:

```
ACCEPT
  IF Security.TakeViewWindowEvent(SSEC::Viewing, ?Cancel)
    CYCLE
  END!IF
CASE EVENT()
  OF EVENT:Accepted
    !DO Something
  END!CASE
END!ACCEPT
```

**Security.Translate ( STRING EquateName ), STRING**

This procedure takes the **name** of an equate from STABSEC.TRN (e.g. 'SSEC::Message:Title') and returns the corresponding text. If it doesn't recognize the equate name, then it checks to see if the first six characters are 'SSEC::', in which case it reports an error. Then it returns the unmodified text (since it cannot provide a properly translated value.)

You can override this function to support multi-languages at run-time.

Example:

```
MESSAGE(Security.Translate('SSEC::Message:AccessDenied'), |
        Security.Translate('SSEC::Message:Title'), |
        ICON:Asterisk)
```

**Security.UpdateActivity**

This procedure is used to track user activity to support the Inactivity Time-out. It updates internal variables used to remember the user's last activity.

Example:

```
Security.UpdateActivity
```

```
Security.UpdateRuntime ( STRING Program, STRING Procedure, <BYTE  
UpdateForm>, <BYTE WatchShift> )
```

This procedure is called to update the run-time security settings for the specified procedure. It is also called by Report procedures to watch for the invocation of run-time security maintenance.

*Program* - The name of the program (without extension)

*Procedure* - The procedure name

*UpdateForm* (optional) - This specifies whether the procedure is an update form (i.e.: a Window with the SaveButton control template). If so, then the user will be allowed to update all the run-time security doors for that procedure, not just "General".

*WatchShift* (optional) - If this is being called by a Report, this parameter indicates that the system should watch if the [Shift] key is depressed. If it is depressed and the user has sufficient access privileges, then the run-time security maintenance window is called for the report procedure.

Examples:

```
Security.UpdateRuntime('Test', 'UpdateEmployee', True)
```

```
Security.UpdateRuntime('Test', 'PrintEmployee',, True)
```

```
Security.UpdateProcedure:Runtime ( BYTE UpdateForm )
```

This procedure is called to update the run-time security settings. It is assumed that the proper SSEC::Procedure record is loaded and that GlobalRequest is set to ChangeRecord. If the operation is completed, GlobalResponse is set to RequestCompleted. Otherwise, GlobalResponse will be set to RequestCancelled.

*UpdateForm* - This specifies whether the procedure is an update form (i.e.: a Window with the SaveButton control template). If so, then the user will be allowed to update all the run-time security doors for that procedure, not just "General".

Examples:

```
Security.UpdateProcedure:Runtime('Test', 'UpdateEmployee', 1)
```

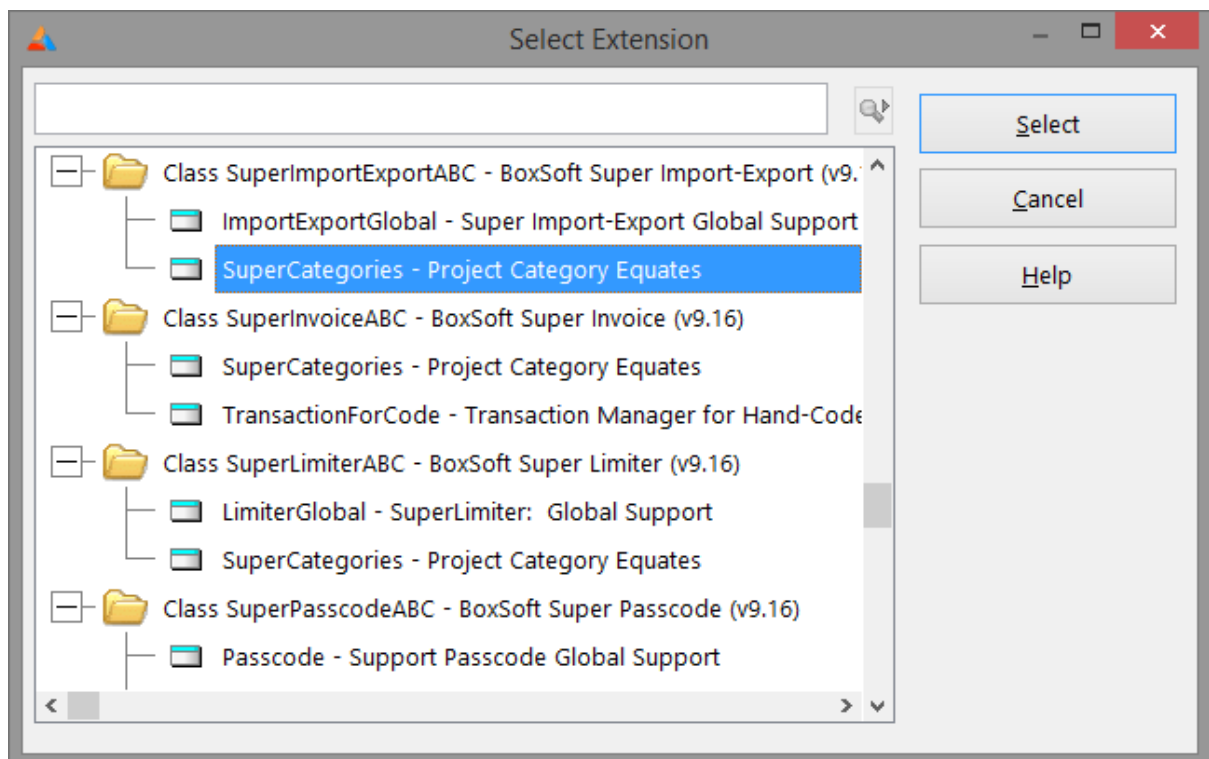
```
Security.UpdateProcedure:Runtime('Test', 'BrowseEmployee', 0)
```

### 3.9 Project Defines

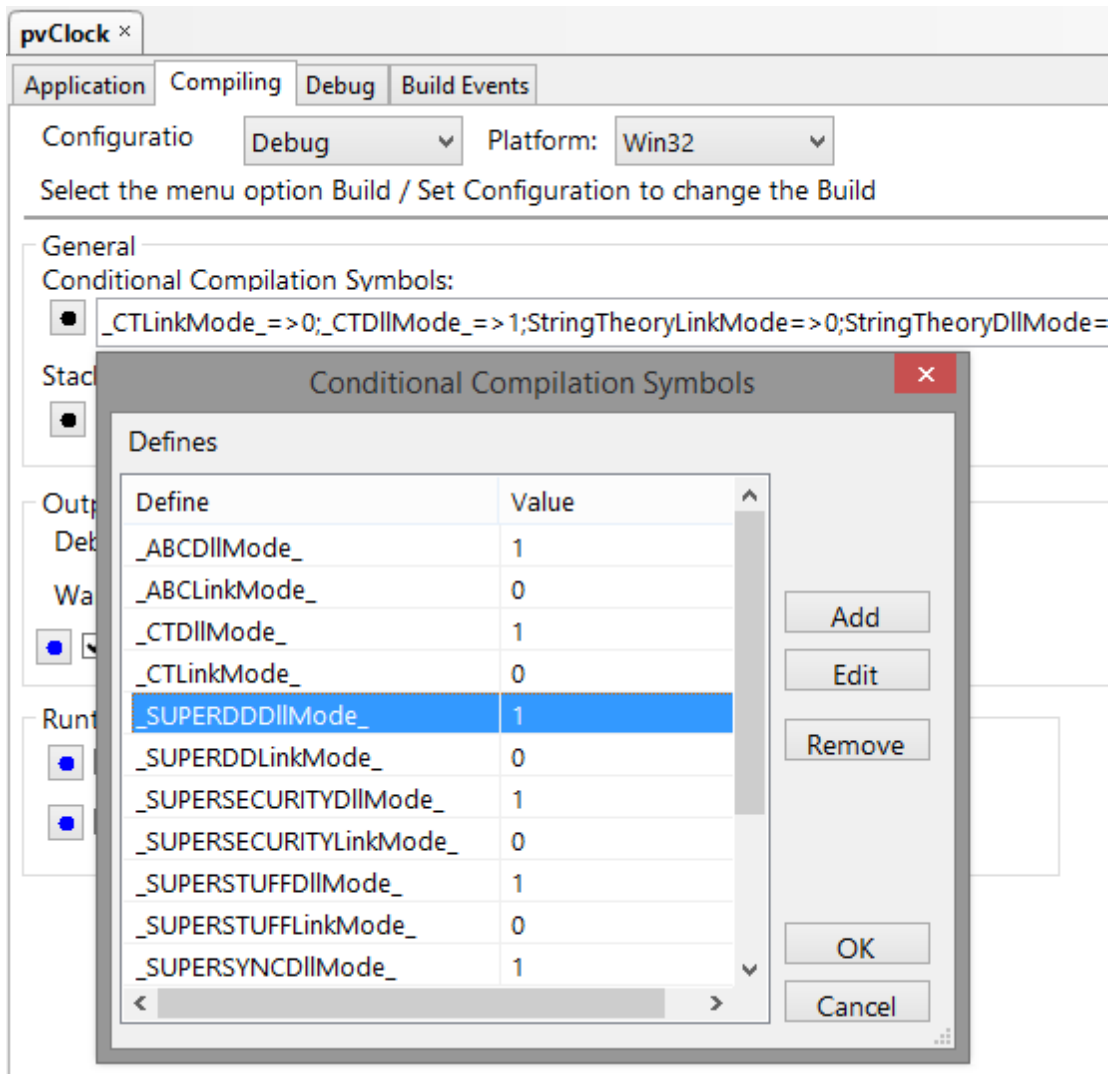
Prior to our 7.0 template versions, we were utilizing the same LINK and DLL Project Defines as the ABC libraries: `_ABCLinkMode_` and `_ABCDIIMode_`. This caused all of our libraries to be included in your base/dictionary DLL, even if you weren't using them in a particular development project.

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 `_SuperQBEDIIMode_`), while others are associated with one of our shared base classes (e.g. Drag & Drop uses `_SuperDDLLinkMode_` and `_SuperDDDIIMode_`). Usually the templates (especially the global ones) automatically add the necessary entries to your Project Defines. If you happen to use the templates in your APPs in the wrong combination, these can be inadvertently omitted.

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



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



## 3.10 Troubleshooting

**Problem:**

You get a compiler error regarding `UserNo_` or `UserName_`.

**Solution:**

Due to a change in version 6.0, we have protected these values within the `Security` class. You must access them via wrapper methods now. For more information, see [Upgrading From Earlier Versions](#) and [API Reference](#).

**Problem:**

You have edited the dictionary, and converted the security files to be stored on an SQL back-end. Now you get an error 33 - *"Record Not Available"*.

**Solution:**

See "Super Security 6.03 to 6.04" in [Upgrading From Earlier Versions](#)..

**Problem:**

You get a compiler error regarding `SSEC: :Logon`.

**Solution:**

Due to a change in version 5.0 (ABC) and 6.1 (Legacy), you must import `LOGON.TXA` into your base (i.e. "Files") APP. This enables your application to be handled by ClarioNet and similar systems.

**Problem:**

Your logon doesn't work, because the files cannot be found.

**Solution:**

For those of you overriding the `PrepareLogonOpenFiles` virtual method, we've changed our approach a little. Now there's a new virtual method called `PrepareFileNames`. It's called before calling the `LoadxxxxQ` methods, so that people with variable filenames can setup these values before Security needs to use the files.

Also, we've moved the loading of the queues from the program initialization to the same location that the Logon is called (in the main menu). If you have significantly tweaked your system, you may have to call `Security.LoadQs` yourself. (It automatically calls `PrepareLogonFiles`.)

**Problem:**

You get error messages pointing to variables called `"SSEC: :Something:Message"` and `"SSEC: :Something:Title"` variables within calls to the `MESSAGE` function.

**Solution:**

All of these variables are defined in `STABSEC.TRN`. You may have copied this file into your application directory, or you may have it somewhere else on your system before the `3rdParty LibSrc` directory.

If you have recently upgraded to a new version of the templates with new entries in the `TRN` file, then you would have to update your application's local copy, or clean-up any unused copies lying

around in the search path.

**Problem:**

You've just upgraded from SuperSecurity 1.9 to 2.0, and now you can't logon using "valid" user names and passwords.

**Solution:**

Did you convert your security data files from the old format to the new?

If the logon window seems to accept the user name, but not the password, did change your encryption mask in the global extension?

**Problem:**

When you execute your EXE, it doesn't display a logon window.

**Solution:**

Is your application's first procedure a Source procedure? The **Security.Logon** call is inserted in the top of the main procedure's **ThisWindow.Init**. If you are using a Source procedure for your main procedure, you'll have to call **Security.Logon** manually.

---

### 3.11 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](http://www.mittensoftware.com)  
[answers@mittensoftware.com](mailto:answers@mittensoftware.com)  
[www.boxsoft.net](http://www.boxsoft.net)  
[www.boxsoft.net/contact.htm](http://www.boxsoft.net/contact.htm)

## 3.12 License Agreement

### **One License per Developer**

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

### **Redistribution**

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

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

### **Disclaimer**

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



# Index

## - A -

Adding Security to your Applications 23  
API Reference 64  
Auditing 4, 12, 24, 27, 45, 46, 47  
Auditing Access to a Procedure 45  
Auditing Field Edits During a Change Operation 47

## - B -

Backdoor 60  
BrowseBox Update Security 37  
Button 48  
Button for Run-time Security Maintenance 48

## - C -

Check for Previous Logon 50  
Class Libraries 8, 75  
Code Template 4, 46, 49, 50  
Conditional Code 49  
Contacting Technical Support 79  
Control Security 43, 44  
Control Template 4, 37, 48  
Creating an Audit Trail Entry 46

## - D -

Defines 75  
Demos 62  
Dictionary 24  
Directories 8  
Disable 60  
Disabling Controls 43, 44  
Disappear 75  
DLL 57  
DLLMode 75  
DoorEdit 52, 56  
Doors 4, 12, 27, 52

## - E -

Embed 46, 49, 50  
Examples 62  
Extension Template 4, 35, 37, 40, 43, 44, 45, 47  
External 27, 57

## - F -

Form Update Security 40  
Freeze 75  
Function Reference 64

## - G -

Global Extension Template 27, 56  
GPF 75

## - H -

Hiding Controls 43, 44

## - I -

Icons 61  
Import 24  
Inactivity Time-out 27  
Include Files 8  
Installation 8  
Interface 61  
Interface Modification and Translation 61  
Internal 27, 57  
Introduction 4

## - L -

Levels 4, 12, 27, 52  
LIB 57  
Library Reference 64  
License Agreement 80  
LinkMode 75  
Logon 27, 60

**- M -**

Multi-APP Development using LIBs/DLLs 57

**- P -**

Procedure Reference 64  
Procedure Security 35  
Project Defines 75  
Protecting Controls on a Report 44  
Protecting Controls on a Window 43

**- R -**

Redirection File 8  
Registering the Template 8  
RTFM Warning 7  
Running a Program to "Check for Previous Logon"  
50  
Run-time Security 4, 12, 24, 27, 48, 56

**- S -**

Samples 62  
SQL 59  
Strings 61  
Support 79  
Support Files and Procedures 24, 56  
Support Programs 52

**- T -**

Tagging 27  
Tech Support 79  
Template Registry 8  
Toolbar 48  
Translation 61  
Troubleshooting 77  
TXA 24, 52, 56  
TXD 24

**- U -**

Update Security 37, 40  
Upgrading 17

User Groups 4, 12, 52  
UserEdit 52

**- V -**

Variable Reference 64