Section 10
Configuration and Sizing

In the following pages, we shall introduce you to the main features of SURE through an explanation of the SURE configuration options. This document describes the theory and procedures behind the SURE system.

Before you start working your way through this document, we strongly advise you to familiarize yourself first with the purpose and terminology of SURE. You will find these explanations in the next section of this tutorial. It may also be helpful to read the Product and Technical Overview of SURE, which you can find in the Unisys eCommunity (http://ecommunity.unisys.com).

10.1. (Configuration) Terminology

Software Configuration Management (SCM)

SCM is also known as Application Life-cycle Management and is best described in the following bullets:

- Storage of all source-files and source related information in a central repository;
- Streamlined and secure separate environments in which all distinct activities of the application life-cycle take place;
- Application of rules and procedures, supported by automated tools through which these rules and procedures are implemented and controlled, and through which the impact of changes on the application life-cycle can be monitored.

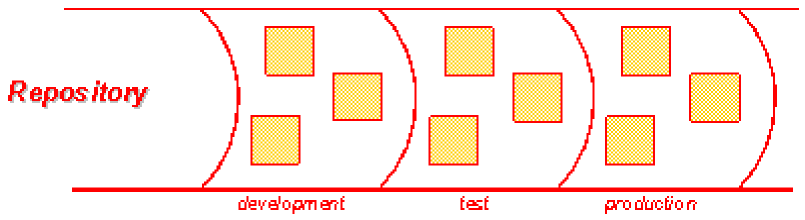
In other words: SCM is an integrated set of tools and procedures that manage/control the software development and the deployment process.

Repository

The repository is a DMSII database on Unisys ClearPath NX/LX or A Series, into which all the source files, delta files containing changes to these source files and other source related information of the various applications an organization has, are loaded and safely stored. The DMSII database provides the advantages of keeping all this information in a single data structure and compresses this information to create a highly efficient storage. The fact that a database is used for source storage also has the advantage of applying proper dump procedures.

Environment

The repository is divided into separate sections, called environments, which match the various phases in the life-cycle support for a particular organization. Typical examples, which are often recognized as separate phases within an organization are 'development', 'test' and 'production'.



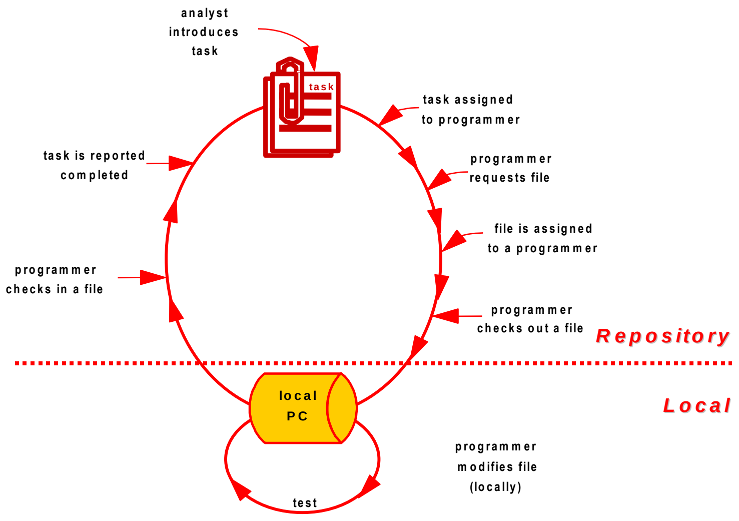
The above given structure may be adjusted to reflect an organization’s own way of working.

Files

For each application system within an organization, all source files are stored in these distinct environments. SURE allows storage of NX/LX source files (regardless of the file-kind), as well as storage of any kind of PC file. Once stored into the repository, files may be retrieved from the repository via a check out procedure and after editing, compiling and programmer-testing, stored back again via the check in procedure.

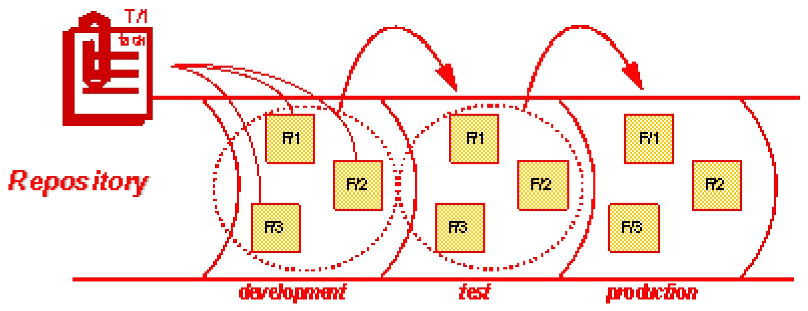
Task

A task is a logical unit of work, and describes the reason of making changes to existing source files or creating new files. The task may apply to a required fix of an error, or to the development of a new feature. SURE links the task automatically to the source files that are modified because of that task. A developer can only check out a source if the task is assigned to him or her. Management assigns this task to one or more developers. Every time a developer starts his or her work, he or she first "subscribes" to the task that is assigned to him. From then on, all the work the developer does in the context of the task will be linked to that task.


analyst introduces task
task
task assigned to programmer
programmer requests file
file is assigned to a programmer
programmer checks out a file
Repository
Local
local PC
programmer modifies file (locally)
test
programmer checks in a file
task is reported completed

Transfer of tasks

The task is the focus point for SURE to keep track of the progress of work. This means that SURE does not allow the transfer (or promotion) of a single source file to another environment. This is always done via the task mechanism. Since a task may involve changes to multiple source files, the transfer of the task ensures that all the changed source files are transferred as one unit of work, for example, from development to test and, after acceptance testing from test to production.



Quick-fixes

When an error occurs in the production version of a program then that program has to be fixed. Usually the program will be changed in the development environment, but it may be possible that the program is already in maintenance in the development environment because of another task, and in that case the fix has to be applied without disturbing the work that is already taking place in the development environment.

SURE offers for this the quick-fix feature: the error is quick-fixed in the current production version of the source, and then applied to the newer versions of the application that are currently prepared in development and in test. The quick-fix facility avoids the re-occurrence of the problem in the future.

If the source where the error must be fixed is not in maintenance in the development or test environments, then the regular development (check-in/modify/check-out/transfer) procedure can be used.

Impact analysis

A key-feature of SURE is the facility of impact analysis. With impact analysis it is possible to determine whether a particular modification in the changed source impacts other source files. A good example of this may be a modification in the source of a single copy-file that is used in a number of programs. SURE knows where this copy-file is used, and SURE will automatically compile all relevant program sources that make use of the changed copy-file. The facility of impact analysis of SURE is continuously present throughout all phases of the development and deployment process, and ensures the integrity of the application systems.

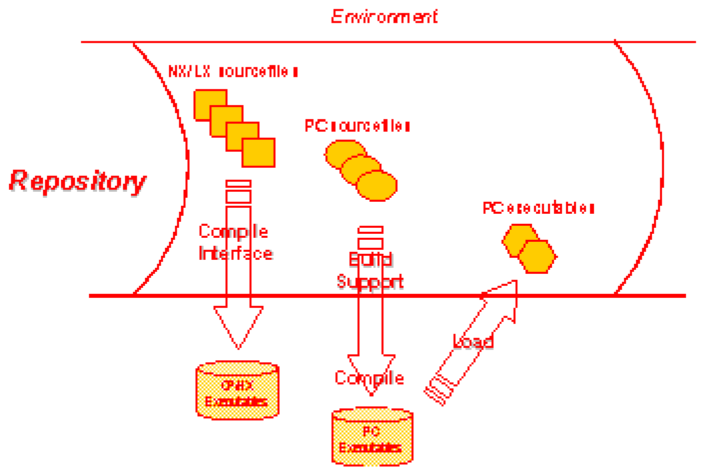
Windows Workbench for LX/NX development

SURE comes with a fully featured development workbench on the Windows platform that works seamlessly with the server-part of SURE running on the ClearPath NX/LX or A Series. The SURE workbench can be integrated with editors, compilers, and any other PC tool from other vendors. This allows developers to download the source files that need to be maintained from the server to a local PC, do all the modifications locally, using their own favorite editor, do even a pre-compile with a local compiler on the PC and upload a syntax-free source file back to the server where it is stored in the repository again.

Build support

SURE supports the automatic compilation of source files of applications for the Unisys ClearPath LX/NX and A Series enterprise servers, such as the compilers for widely used programming languages as Cobol, Algol, Dasdl and WFL, but also less commonly used compilers such as Pascal, Fortran77 and C++.

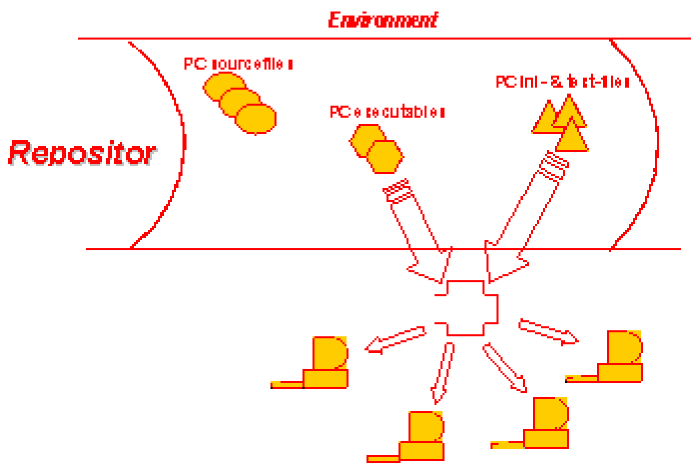
SURE provides build support for the Unisys NX/LX platform as well as for the Windows platform. On the Unisys platform, this is achieved in a direct way, by compiling the sources directly for proper development phases for which a separate environment is kept. Through the task mechanism of SURE supported by its impact analysis facility, the compile process is managed automatically. The resulting object-files are kept outside of the repository; however relevant information of the object-files (e.g. versioning) is stored.



For the PC source files SURE supports the building of the necessary files for the compilation process that takes place on the Windows platform. After compilation, the resulting executables are loaded into the SURE repository as well.

Distribution support

The distribution support for ClearPath NX/LX series allows copying object files to the defined server machines using BNA transfer. The distribution support in the Windows environment consists of preparing a directory that needs to be distributed. A distribution tool like Microsoft SMS or a tool like RoboCopy performs the actual distribution.



10.2. (Configuration) Quick Overview to SURE

Before we start to have a closer look at SURE, we shall first give a short overview in broad terms of SURE and its architecture.

SURE provides a comprehensive tool-set for SCM that makes it possible for an organization to manage and control all phases of the development and deployment process of applications.

SURE manages the repository on the Unisys ClearPath NX/LX or A Series, in which all source files of the organization’s applications are stored.

SURE keeps a constant eye on the changes made to the contents of the repository and takes automatic actions accordingly, such as (multi-)compiles of the changed source files, transfer of new versions of an application from development or test to production, etc.

SURE keeps a strict separation between work that is done during the actual development or maintenance processes and work, which is required during production, such as error fixing. To this extent, SURE maintains the environments, in which distinct versions of entire applications are stored.

All activities during development and maintenance are carried out under the umbrella of the task mechanism in SURE. This adds significant value to the development and maintenance process in terms of completeness of work and integrity of the applications.

SURE consists of two parts, one part running on the ClearPath NX/LX or A Series that effectively manages all aspects of the repository, does the impact analysis and provides compiler support. The other part of SURE, being the client-part, is a fully featured development workbench on the Windows platform that works seamlessly with the server-part.

SURE provides an organization with the means to control all of its applications, whether these exist on the Unisys platform or on the Windows platform, in an integrated manner, thus ensuring the quality and integrity required to support today's e-business requirements.

SURE can work in three modes:

- Source-maintenance:

  The stable version of each source is stored in the SURE repository. A developer uses function 'check-out' if he wants to modify a source and function 'check-in if the modifications are done. SURE keeps a physical history of each source.

- Source-maintenance + tasks & environments:

  All functions of 'source-maintenance' plus an integrated task tracking mechanism. A task describes the reason why a source has to be modified, and this task is linked to the source when that source is checked-out. The (source-modifications because of a) task can be tested in different environments (TEST, ACCEPTANCE, et cetera). SURE keeps also a logical history of each source

- Source-maintenance + tasks & environments + application deployment:

  All functions of 'source-maintenance + tasks & environments' plus an integrated application deployment mechanism. Files that are checked-in to SURE are automatically compiled (during an evening batch). If a copy-file is changed, then the calling programs are compiled. The location of the compiled object can be defined in SURE. SURE copies the compiled objects to the correct object locations. SURE guards the integrity of the total object environment.

10.3. (Configuration) SURE Configuration

SURE is a source management system, as well as a system that provides software build, configuration and distribution support. Because of the support for these integrated software life-cycle functions, the SURE system allows for simple configurations with just check-in and checkout support up to complex configurations, which include support for the creation of executables and distribution of software. This document guides you through the configuration options and it explains the various supported procedures.

The SURE system supports UNISYS CP/NX files, A-Series files, Windows-32-bit files and UNIX files, if they are shared from a Windows system. This document applies to all of these mentioned types of files unless it is explicitly stated otherwise.

The SURE system contains of two different parts, a server part implemented on the UNISYS CP/NX or A-Series, which is used as a storage server. This server part consists of a DMSII database with a software layer around it, which let the server part behave as a repository. This DMSII database will contain all of your definitions and sources so that a simple database dump procedure can be used for archive purposes. The second part of the SURE system consists of a 32-bit Windows client software package, which provides a user interface similar to Windows Explorer. Server and client part are connected through native TCP/IP, which allows you to run over a LAN or over the Internet.

The SURE system allows for a dynamic configuration. This means that the SURE system may be tailored to the specific procedures within an organization. Because the SURE system is flexible in its configuration, it requires that the system must be configured.

Configuring the system is not a difficult task if you are familiar with the capabilities of SURE. However, it does require you to make decisions based on the capabilities of SURE in relation to the procedures currently applied within your organization. Therefore, configuring the system can be challenging if you are not (yet) familiar with the capabilities and concepts of SURE.

Another complicating factor is that you might want to change currently applied procedures because SURE offers other capabilities. It is obvious that some of your current procedures must be changed when you are going to use SURE (e.g. your current procedure 'assign a source to a developer' will change because that source is now loaded in SURE). Others of your current procedures can operate alongside SURE.

SURE is capable to control up to eight different environments (design, develop, test, integration, production, history, etcetera), each with their own set of objects in a separate run-time environment. Run-information about the objects can be stored into SURE. SURE has a flexible and powerful authorization mechanism that can limit the programmer's activities. SURE can address various modes of working, such as: shared workspaces, individual workspaces, baselines, and etcetera.

However, you do not have to use all functions of SURE, and you do not have to activate all desired functions at the beginning when SURE is installed.

From our viewpoint, we would like to stress that a new tool and new procedures introduce an additional risk and this may decrease the acceptance level within an organization. For this reason we advise to change procedures gradually afterwards if that is possible. In this way the users are used to the tool SURE when a procedure changes.

The SURE system allows you to define rigid procedures or it allows you to define flexible procedures. For example: SURE allows you to define that a programmer first requests a file, after which the project leader releases the file for this programmer allowing the programmer to checkout the file. This example of a rather rigid procedure is fully supported by SURE with the appropriate commands and workflow queues. However SURE also supports a procedure where the programmer directly checks out the file for a specific task. This is a more flexible procedure, but it requires programmers that are more experienced.

10.4. (Configuration) Goal

This section contains background information, which allows you to configure a repository, load files in this repository and finally perform lifecycle support on these files.

10.5. (Configuration) Functionality

SURE is a task-based system, which requires defining a task for a functional change to an application system. All the functional changes for this task, to different files that may reside on different platforms, are connected to this task. This task or logical unit of work is taken into acceptance or into production. SURE supports this procedure by workflow queues, task distribution over different employee functions and support for the build and distribution process.

A task is a logical unit of work that matches a functional requirement. SURE offers functionality to group tasks together, so that those tasks are released at the same time. A task can also be used as a release task: all the changes for a specific release, carried out by different programmers, are grouped under this task and deployed as a new release.

SURE supports the implementation phase of software construction. This support deals mainly with source maintenance, which is referred to as life-cycle support. This life-cycle support deals with the simple straightforward procedures like check-in and checkout and a role based security mechanism. On the other hand SURE offers work flow queues for the various employee roles such as a To Do list for a programmer or a To Be Accepted list for quality assurance employees or a Ready and Busy list for a project leader.

These lists are present in every fixed stage into your source life-cycle. Within SURE, such a fixed stage into your source life cycle is called an environment. Furthermore: these lists may be distributed over employee roles and filtered for specific application systems or sub-systems.

Within the life-cycle support functions, the quick fix procedure and the impact analysis are main sub functions.

The quick fix procedure allows you to make fixes to your production software without disturbing the development process. Even if the file you are fixing is currently in maintenance by another programmer. Firstly, the SURE software will prohibit accidental overwriting of this fix, so the error cannot appear again in a new release by accident. Secondly, the SURE software offers automated functionality to incorporate the fixes into the standard development process.

The impact analysis allows you to detect whether different functional changes resulted in modification of the same file and it may provide you with a compile impact, a list of programs that need to be compiled for a change. This compile impact includes manipulation of copy or include-files. If a copy or include file is changed, SURE will recompile all the programs using this changed copy or include file.

Compilation and build support is an optional feature that is supported by SURE. This feature creates the executables from the CP/NX sources as well as for PC applications. For CP/NX software the compilation support consists of a program that compiles all the changed files with the appropriate compiler. The repository contains definitions for file security, file attributes, task attributes, binding and post compilation requirements. For CP/NX and A-Series files, the resulting object is given a unique release id, which matches the current file version of the compiled source. An especially useful command for CP/NX and A-Series is ‘recompilation of sources using a specific dataset’. Reorganizing a database always requires such a procedure, and this procedure is smoothly integrated in the SURE software. For PC and UNIX files, the repository contains the build commands. These build commands can be seen as a .BAT file which contains the appropriate commands that are required to create the executable. The build support for PC and UNIX files downloads all the changed files to a server, secondly it executes the build commands per file or per directory and in the final stage it loads the resulting executable into the repository.

Distribution support is an optional feature that is supported by SURE. Distribution support actually copies the files to the target destination. For CP/NX and A-Series files, this may be a user code and pack family on the same host or on another BNA host (tape transfer is still supported for

CP/NX and A-Series systems, but it is not the default method anymore). For PC and UNIX files, a list of files is maintained of the actual files that need to be distributed. This list can be downloaded in a directory. The SURE system supports creation of this directory. Other sub systems must be used to perform the actual distribution. An example distribution system is Microsoft SMS.

The minimum configuration for SURE is using a simple form of life-cycle support. Using SURE in this fashion return direct benefits like: who changed what, where and when. Secondly, you can view the file changes and rebuild any previous version. In this mode, a programmer can reuse his task repeatedly.

The next stage in using SURE consists of defining your stages in a source life-cycle. These stages (environments) will allow you to use most of the life-cycle support functions including workflow queues.

The final stage in using SURE implies using build and distribution support. Hereafter you can run your system with only a single person controlling it. If the SURE build and distribution support functions are used, your total operation is automated and no manual actions are required.

10.6. (Configuration) File Entities

- Environment
- System
- Project or sub-system
- File Type
- File

All definitions for files are assembled using the previously mentioned entities.

- An environment is a fixed version in the life-cycle of a file. For example: the test-versions of all files form together the test-environment.
- A system is a set of files grouped together based on physical coherence, so all files using the same database might belong to a system. At system level, different physical attributes are defined which apply to all the files of that system. A system always contains at least one project (sub-system): a project with the same name as the system name. On the other hand: a system may contain multiple different other sub-systems (projects).
- The word ‘project’ can have two meanings:
  - A sub-system
  - An amount of work to be done.

  In the SURE context, the word ‘project’ means ‘sub-system’.

  ‘An amount of work to be done’ is in the SURE context ‘a task’.

  A project is mostly used for security reasons. Persons may be granted to perform functions on files of one or multiple projects. From a logical viewpoint, projects are a collection of files that are grouped functionally. The system level is mandatory within the SURE system, however a detailed project level is obsolete in which case the system name equals the project name.

- A file-type categorizes the files for SURE so that the same definition may apply to a group of files. A file-type is a rather abstract definition that contains attributes such as:
  - A file of this category creates delta files
  - A file of this category needs to be compiled
  - Etcetera

  Therefore, a file-type categorizes files in such a way that SURE performs certain actions on those files. Notice the difference with the contents dependent file-kind attribute such as Algolsymbol or C files. Both of these files can be split over two different SURE file-types, the first describing that it is a include file (CP/NX or A-Series) or a header file (PC or UNIX) and the second describing that it is a source which needs to be compiled.

10.6.1. (Configuration) Environment

A key entity in the SURE system is an environment, which can be considered as a fixed phase in a development life-cycle.

An environment is a fixed phase in a development cycle. Common names for environments are DEVELOPMENT, ACCEPTANCE or PRODUCTION.

In this example:

- DEVELOPMENT defines the phase where programmers adapt files for implementation of new functionality. Module testing by a programmer is included in this phase.
- ACCEPTANCE is a phase where quality control performs integrated test scripts on the implemented functions. Therefore, the quality control accepts functional changes and marks them as ready for production.
- PRODUCTION is the phase where the actual software runs on the various production systems.

Another company may have the same phases in their development as the previous example, but they also require a hardware integration test. This company runs the application system on different hardware using different compilers and they require a final test on the production hardware before it is actually taken into production. This company calls this environment BETA_SITE. Therefore, the environments for this company are called DEVELOPMENT, ACCEPTANCE, BETA_SITE and PRODUCTION.

Above given example defines a software life-cycle often used in companies writing their internal software. Mostly these companies do not support official releases of application packages; however this is a default requirement by software manufacturers. The previous example shows a production environment that contains all the files required to support the production application systems of the company.

If we would consider a software manufacturer, then the environments would be used more dynamic. Again, one could support DEVELOPMENT and ACCEPTANCE as their lowest developments environments. The development process and quality control may be performed in the same sequential flow. However, after acceptance, multiple different release environments may be present. So after the environment acceptance a RELEASE_5.0 environment may be present and thereafter RELEASE_4.0 and thereafter RELEASE_3.0. A software manufacturer often supports multiple different releases because each release may still be in use by one of his customers for which he agreed a support contract. A customer running software release 3.0 may require patches that must be created via software updates on release 3.0. Another customer running on release 4.0 requires updates for that release.

The software manufacturer has the two lowest environments (development and acceptance) for his own life-cycle purposes. The number of environments after acceptance and their names may vary. If an old release is not supported any more, then the environment is deleted. If a new release is created, then a new environment for that release is created. Therefore, when release 6.0 is created, a new environment is created called RELEASE_6.0 directly above the acceptance environment. The content of this RELEASE_6.0 environment is copied upwards from the acceptance environment.

Now let us have a more detailed view on the contents of such an environment. First, each environment contains all sources. Each environment has a corresponding compile & build process and that process uses the sources from that environment as input to create the object files & executable files. Additional definitions that are used to create the executables (such as task attributes, file attributes, translate tables and build commands) are stored in the repository. The repository also contains definitions that support the distribution process (object locations).

As described, an environment contains all the sources, however for PC files it will also contain the resulted executable. This is not true for CP/NX and A-Series files where the executables are not loaded in the repository.

For maintenance reasons the definitions for build and distribution support are usually not defined at individual file level. There are three entities that contain most of the build and distribution definitions: file-type, system and project.

An environment contains all files. Each environment can contain another version of a specific file, but a specific file-version can also be in use in multiple environments. The files are grouped in application systems and a release notion can be defined for a system. Therefore, an environment contains multiple systems and a system contains a set of files for a defined release. An environment contains a release for each defined application-system.

10.6.2. (Configuration) System

System is the most important configuration entity.

A system is a set of files grouped together because they share the same physical attributes. In the SURE software, this applies to the attributes work-location and object-location. These work and object characteristics are defined per environment.

10.6.2.1. (Configuration) Work Location and Object Location

The work-location is only applicable for CP/NX files and it designates the directory (usercode and the pack name) where the shared resources are located. With shared resources, we mean copy files or include files, a database description file, and etcetera.

The work location is used for two purposes:

- The work location defines the shared resources directory.

  Shared and stable resources (copy files, description file, etc.) are used by programmers when they do a test compilation of a source that they have in maintenance. It is obvious that the version of a copy-file can change per SURE environment. The programmer expects that the correct version of each copy-file is available on disk in the shared resources directory, otherwise he cannot compile his source correctly.

- The work location determines the programmer’s work directory.

  The programmer has to be logged on to this directory when he wants to work on a task (updating sources that are loaded in SURE). The sources-in-maintenance are placed in this directory.

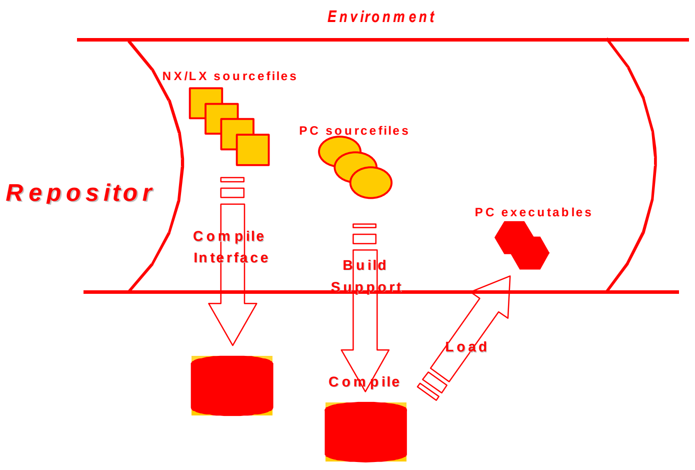
If a file has an object-location then it can be deployed. The object-location of a CP/NX file determines the destination directory (usercode, pack, and host) for the resulting object after the source is compiled by the SURE batch. The object-location of a PC/UNIX file determines the logical server name where the build process is done.

The object-location of a system is the default object-location. When a new file is added for that system, then that file inherits the default object-location.

The object-location is applicable for both CP/NX files and PC/UNIX files, but the implementation for both kinds is different.

- For CP/NX files the object-location designates the default directory (usercode, pack name and optionally the host name) for the resulting objects after they have been compiled by the SURE build support. In other words: the object destination for the distribution support.

- For PC/UNIX files, the object-location is a logical server name, which is assigned as the build server. In other words: the executables for this system are built on the assigned server. Therefore, for PC/UNIX files, the object-location defines the server where the actual build process is executed. The build process copies all the changed files (with that build server) to the configured directories and start the build commands. Therefore, you can just define the same build server for every environment if you configured different directories per environment for this server.



Summarized:

The work-location of a system affects the programmer’s way of working because it defines the directory where the shared resources are placed, and it determines the directory where the developer must be logged-on to be able to do his job. The object-location of a system is only a default value for the object-location of a file and it does not affect the programmers’ way of working.

From the previous description, you might conclude that a system is merely a technical separation. However, in most circumstances a system is defined as an application system such as a mortgage system, a credit card system, a stock exchange system or a general ledger system. This is because all of the files within such a system normally share the same work and object locations.

The SURE manual (chapter Copy Files) goes into detail on all the different options that influence the work-location, including a rather complicated option called ‘resources’.

A SURE repository is sub-divided into multiple environments (for example: develop, test and production). A file is modified in the develop environment, and transferred from develop to test and production. If a file is transferred to production then it is equal in all environments, until somebody makes another modification to that file in the develop environment.

Notice again that a stable file is equal in all environments, and therefore it is important that all usercodes and packnames are removed from the sources that are loaded in SURE (because production usercodes/packnames are usually not the same as test or develop usercodes/packnames).

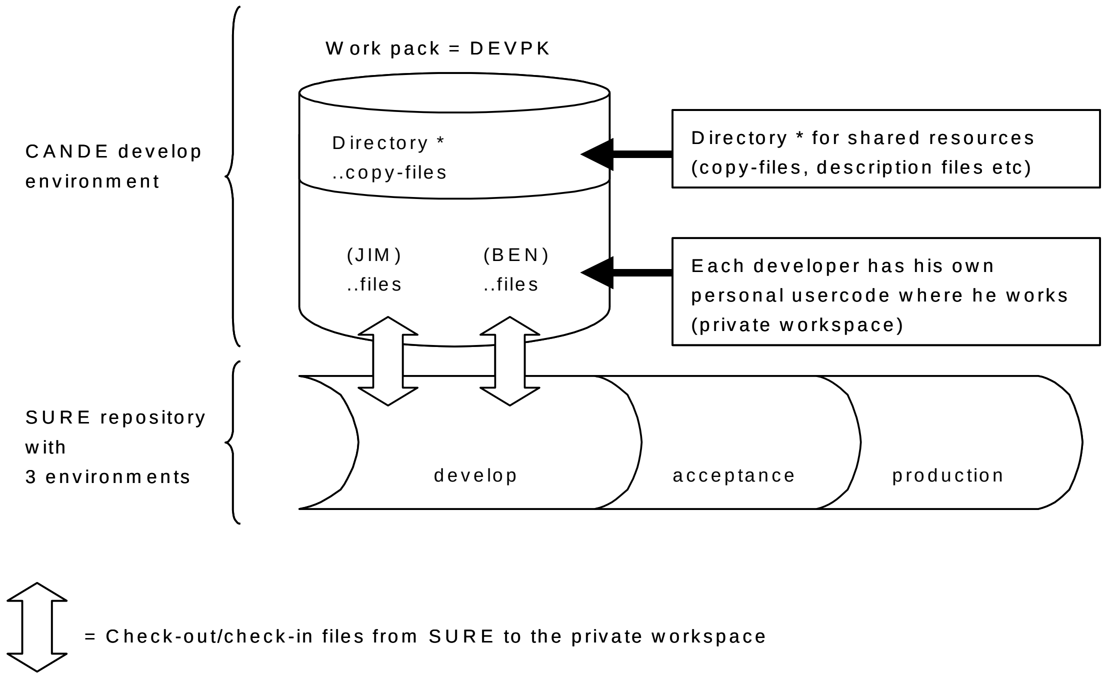
The following paragraphs explain some basic configurations on which you have to decide. These configurations depend basically on the definition of the work-location.

Notice again that the work-location is defined per system, and that makes it possible to use a different configuration for each application system.

10.6.2.2. (Configuration) Private Workspace / Shared Resources

Characteristics:

- Each developer has his own personal usercode to log on.
- Each developer has his own personal workspace where he does his work (editing, compiling, and testing). This personal workspace is identified by his personal log-on usercode.
- The shared resources are placed in the global directory * (without a user code) on the work pack.


Work pack = DEVPK
CANDE develop environment
Directory * ..copy-files
Directory * for shared resources (copy-files, description files etc)
(JIM) ..files
(BEN) ..files
Each developer has his own personal usercode where he works (private workspace)
SURE repository with 3 environments
develop
acceptance
production
= Check-out/check-in files from SURE to the private workspace

The sources don't contain any usercodes and packnames.

A programmer logs on with his personal usercode and checks a file out of SURE. The source is placed on disk under his personal usercode. The copy-files that are used by the source must be resident on disk, otherwise it is not possible to compile that source.

The copy-files are resident in the shared resource directory (* on the work-pack) and these file are available for the developer via the standard visibility rules of Cande: if a copy-file is not resident under the developers' personal usercode, then that copy-file is found in the shared resource directory.

If a copy-file is checked-out from SURE, then that copy-file is also placed on disk under the developers' personal usercode, where the developer modifies that copy-source. The modified version of the copy-file is only visible for the developer who checked the file out. The other developers (who are logged on with other usercodes) still use the stable version of the copy-file from the global directory *.

If the new version of the copy-file is checked-in to SURE, then it is removed from the developers' personal workspace and copied to the shared resource directory (* on work-pack) where it is visible for the other developers.

An advantage of this option is that other users are only aware of the modification of a copy-file after that copy-file is checked in. A developer changes a copy-file in his own private workspace, and the test results of other developers are not affected by the changed copy-file.

Mode 'Private workspace with shared resources' can be defined with the following system options:

- The 'shared resources' functionality is established by option 'Put changed copy-files in work-environment'.
- The 'private workspace' functionality is established by: Work-location-usercode = *

Example

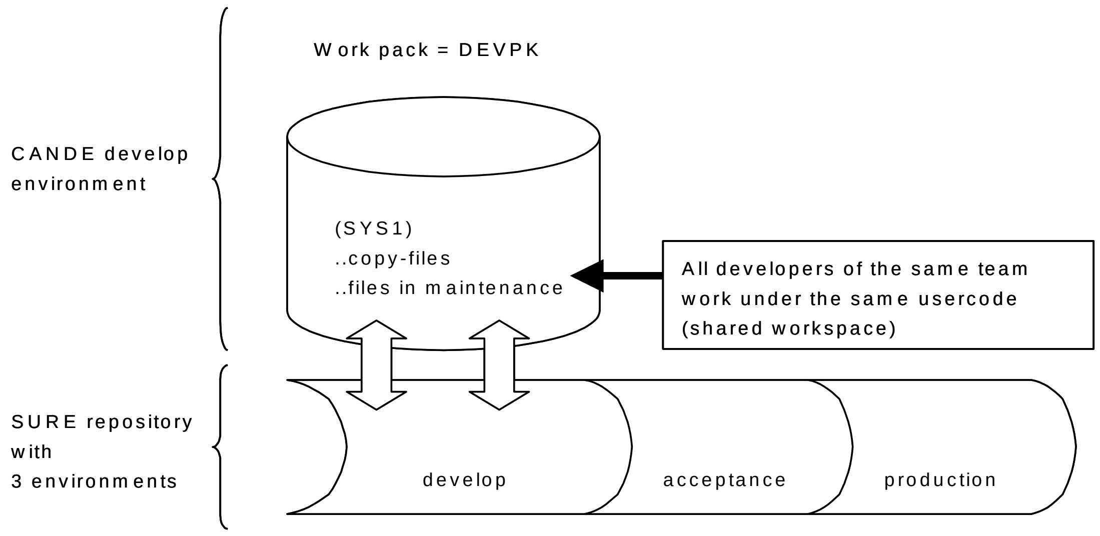
Consider copy-file SYS1/COPY/001.

- Version 3.1 of that copy-file is active in all SURE environments and that version is also available in the shared resource directory: *SYS1/COPY/001 ON DEVPK.
- Programmer Jim does a check-out of the copy-file and the source is copied from SURE to his private workspace: (JIM)SYS1/COPY/001 ON DEVPK
- Jim modified the copy-file in his private workspace. Test compilation started by Jim use the copy-file in his private workspace, because the source that references the copy-file does not contain usercodes/packnames in the copy-statements.
- Programmer Ben uses all the time the stable version 3.1 of the copy-file.
- Jim finished his work on the copy-file and does a check-in. The new version of the copy-file is loaded in SURE with version number 4.1; the copy-file is removed from Jims' private workspace and copied to the shared resource directory.
- From now one version 4.1 of the copy-file is used by all developers.

10.6.2.3. (Configuration) Shared Workspace / Shared Resources

Characteristics:

- All developers modify their sources in the same workspace (usercode/pack).
- The source files and copy-files are maintained under that same usercode.
- The shared resource files (copy-files, description-files) are placed on disk under that same usercode.


Work pack = DEVPK
CANDE develop environment
(SYS1)
..copy-files
..files in maintenance
All developers of the same team work under the same usercode (shared workspace)
SURE repository with 3 environments
develop
acceptance
production

The main differences between this mode and the previous mode (private workspace with shared resources) are:

- The user does not have an individual work environment. So, during the modification of files, every other user may be affected during the module test.
- All copy-files are always available in the shared workspace. That makes it possible to change a copy-file without officially checking it out from SURE. This situation is not critical for the production environment because such a file cannot be checked-in, but it may cause temporary questions (who changed his source and why did he do that?).

Each developer still logs on to SURE with his own personal usercode. This log-on usercode is only used for identification purposes. If the developer checks a file out, then that file is placed in the shared workspace.

Mode ‘Shared workspace with shared resources’ can be defined with the following system options:

- The ‘shared resources’ functionality is established by option ‘Put changed copy-files in work-environment’.
- The ‘shared workspace’ functionality is established by: Work-location-usercode = <usercode>

Example:

Consider the shared workspace (SYS1) ON DEVPK

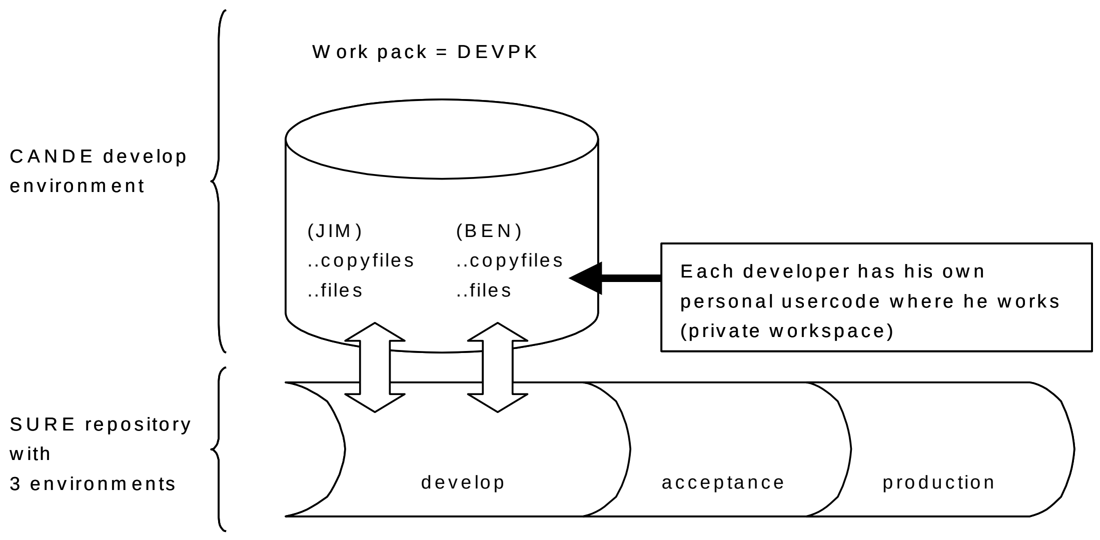
- User Jim logs on to SURE with his personal usercode JIM
- He checks a file out of SURE, and the file is placed on disk as (SYS1)SYS1/PROG/001 ON DEVPK.
- He checks a copy-file out of SURE. The copy-file is placed on disk as (SYS1)SYS1/COPY001 ON DEVPK.

- The source and copy-file are now visible for all other team members that work on system SYS1.

10.6.2.4. (Configuration) Private Workspace / Private Resources

Characteristics:

- Each developer has his own personal usercode to log on.
- Each developer has his own personal workspace where he does his work (editing, compiling, and testing). This personal workspace is identified by his personal log-on usercode.


Work pack = DEVPK
CANDE develop environment
(JIM) ..copyfiles ..files
(BEN) ..copyfiles ..files
Each developer has his own personal usercode where he works (private workspace)
SURE repository with 3 environments
develop
acceptance
production

A checked-out source is placed in the private workspace of the developer. All copy-files that are referenced by the source are also copied in the same private workspace. When the source is checked-in then the copy-files are checked and cleaned-up (copy-files that are not referenced by other checked-out sources are removed from the private workspace).

This mode is defined with the following system options:

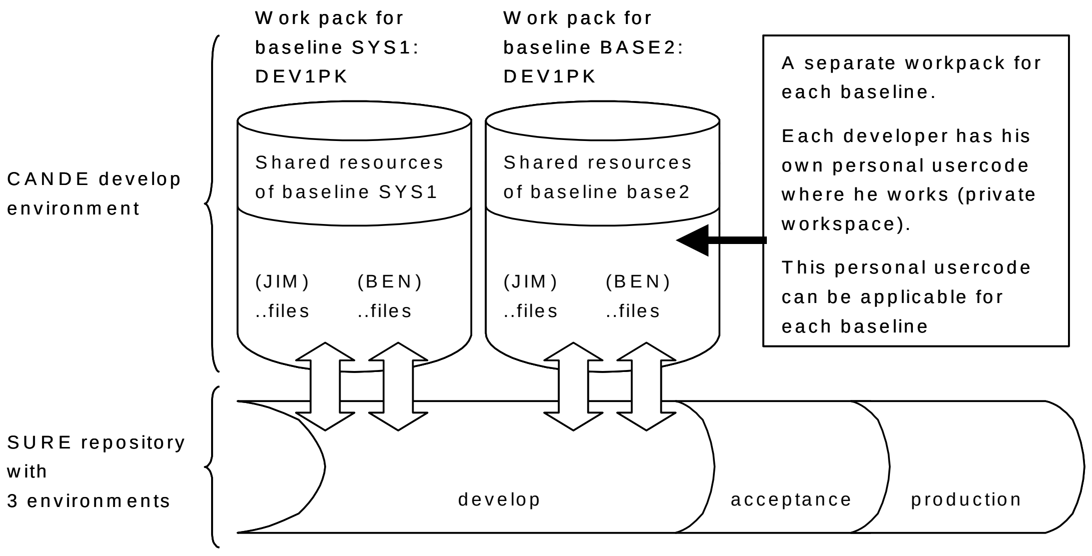
- The ‘private resources’ functionality is established by resetting option ‘Put changed copy-files in work-environment’.
- The ‘private workspace’ functionality is established by: Work-location-usercode = *

10.6.2.5. (Configuration) Multiple Baselines

Characteristics:

- Each developer has his own personal usercode to log on.

- Each developer has his own personal workspace where he does his work (editing, compiling, and testing). This personal workspace is identified by his personal log-on usercode.
- The total development staff that works on the same application system is subdivided in several development teams.
- Each team works on a separate big task (also known as a baseline)
- Modifications that are made for a specific baseline may not be visible for other baselines unless the project leader of the other baseline explicitly accepts these modifications.


Work pack for baseline SYS1: DEV1PK
Work pack for baseline BASE2: DEV1PK
A separate workpack for each baseline.
Each developer has his own personal usercode where he works (private workspace).
This personal usercode can be applicable for each baseline
CANDE develop environment
Shared resources of baseline SYS1
Shared resources of baseline base2
(JIM) ..files
(BEN) ..files
(JIM) ..files
(BEN) ..files
SURE repository with 3 environments
develop
acceptance
production

Each baseline is linked to a separated work pack.

If option ‘put changed copy-files in the work-environment’ is set for a specific baseline, then that baseline works with shared resources.

If option ‘put changed copy-files in the work-environment’ is reset for a specific baseline, then that baseline works with private resources.

If Work-location-usercode = * for a specific baseline, then each developer of that baseline has a private workspace.

If Work-location-usercode = <usercode> for a specific baseline, then all developers of that baseline work under the same shared workspace.

A patch (modification) to a source is done within a baseline, and the name of the baseline is linked to that patch. Patches of a specific baseline are not visible (in copy-files or in compiled objects) for the other baselines, unless the project leader of another baseline explicitly accepted the patch.

The source can be checked-out for two different baselines at the same time.

The patches of all baselines are all merged into the source in the first environment where the baselines are not defined.

Example:

The objects on pack DEV1PK are created from the development environment in SURE and contain only the patches of baseline SYS1.

The objects on pack DEV2PK are created from the development environment in SURE and contain only the patches of baseline BASE2.

The objects that are created from the acceptance environment in SURE contain the patches of both baselines SYS1 and BASE2.

Extra baselines are defined via system option ‘Is baseline of’.

Example:

Consider system SYS1 and an extra baseline for that system BASE2

BASE2 is an extra baseline of SYS1: BASE2 is baseline of SYS1 with work pack DEV2PK

SYS1 must be a baseline of itself: SYS1 is baseline of SYS1 with work pack DEV1PK.

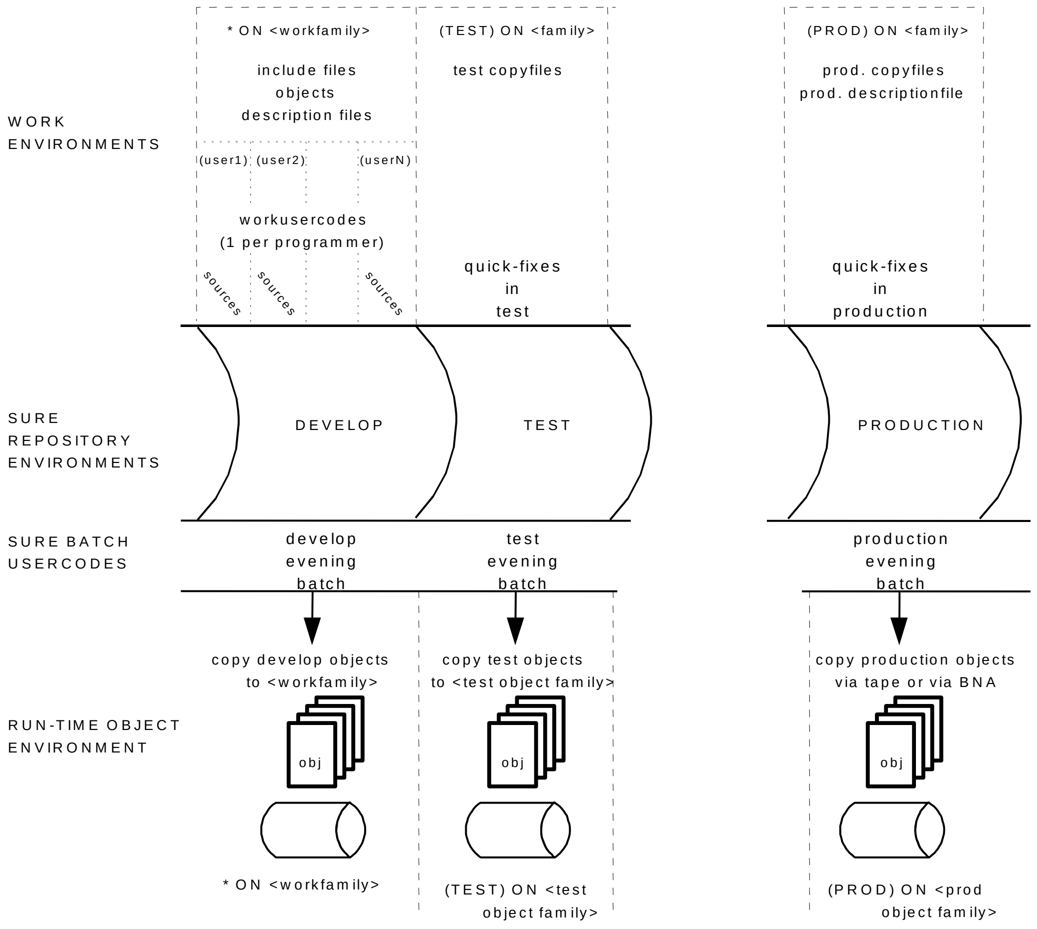
10.6.2.6. (Configuration) Hybrid Configuration

Many complex configurations are grown from using own tools and procedures. This may imply generation of jobs with different object names for test and production, replace utilities that replace the copy or include statements for test and production, etc.

The work environment and object environment are used for the build support, distribution support and the workspace of the programmer. You can always use SURE for check in and check out purposes in combination with your current procedures. In this mode, you do not use the automated mode for build and distribution support and you only use part of the life-cycle support. However, you still can use many other SURE features like the auditing, log, delta files, PC editing, impact analysis etc. Using this start, you could convert different systems gradually if required.

10.6.2.7. (Configuration) Summary

The following picture shows a develop environment with shared resources and private workspaces, and test and production environments with shared resources and a shared workspace.



The following options are applicable in the discussion of workspaces:

- File Type: Use as Copy file (SURE manual chapter 32)

  Files that are actually referenced by other programs as a copy-file behave automatically as a copy-file (and will be placed in the shared resource directory if option ‘put copy-files in the work-directory’ is set).

  It is possible to link this copy-file-behavior to other files via file-type option ‘use as copy-file’.

- System: Put Copy files in the work directory (SURE manual chapter 14)

  This option puts each copy-file and each file that behaves like a copy-file in the shared resource directory after it has been changed.

- System: Work Environment (SURE manual chapter 14)

  This option defines the shared resource directory

Workspace	Resources	Put Copy in Work	Work User Code
Private	Shared	Y	*
Shared	Shared	Y	<usercode>
Private	Private	N	*

10.6.3. (Configuration) Project

A project is a sub-system.

The project definition within the SURE system is optional. When you have created a system, a project with the same name as the system name is implicitly defined within the SURE system. So effectively, you can have an installation without specific projects defined. From a hierarchical point of view, a system can have multiple projects and a project can have multiple files. However, a file or a task can only belong to a single project.

Example:

Consider system SYS1 with project PRJ1 and system SYS2 without specific projects.

A system is always a project of itself, so the following system/project combinations are available:

System	Project
SYS1	SYS1
SYS1	PRJ1
SYS2	SYS2

If a file is added for project PRJ1 then it is automatically linked to system SYS1.

If a file is added for project SYS2 then it is automatically linked to system SYS2.

It is not necessary to define specific projects. Each system is always available as project, and if the system and project are always the same then there is no need for extra projects. On the other hand, a project may be used just to define new tasks. There is an automatic naming standard where the name of a new task is based on the project name. This makes it possible to identify a task by its names as being a task of a certain project.

The major functions that apply to projects are:

- Security (SURE manual chapter ‘Authorization mechanism’):

  The security map is defined per employee function. This is commonly seen as ‘role based security’. The granularity of a security map can differ. It can be global and then it applies to all environments and to all projects (who can do what?); or it can be specific and it applies to a specific environment (who can do what in which environment?) and/or to a specific project (who can do what in which environment for which group of files?).

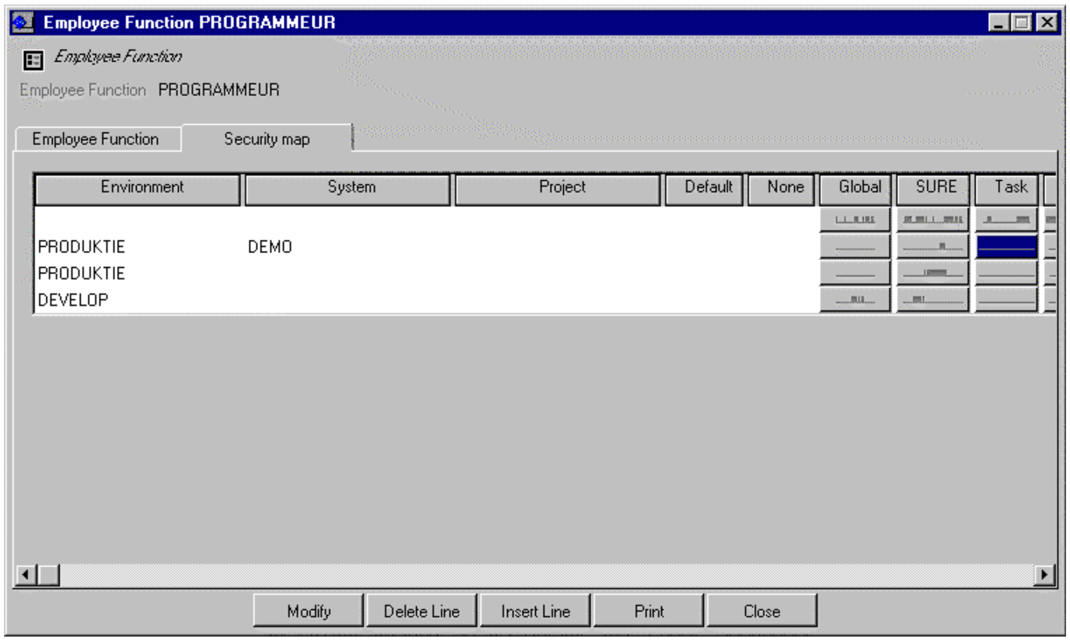
  It is possible to define securities global, per application system and per project. The most detailed level of security definition is at project level.

– Global securities apply to all systems (and all projects).

– Securities that are defined for a system apply to the system itself and to all projects of that system.

– Securities that are defined for a project apply only for that project.

The example screen shows a security definition for an employee function programmer who has different securities for DEVELOP and PRODUKTIE and in PRODUKTIE extra securities for system DEMO.



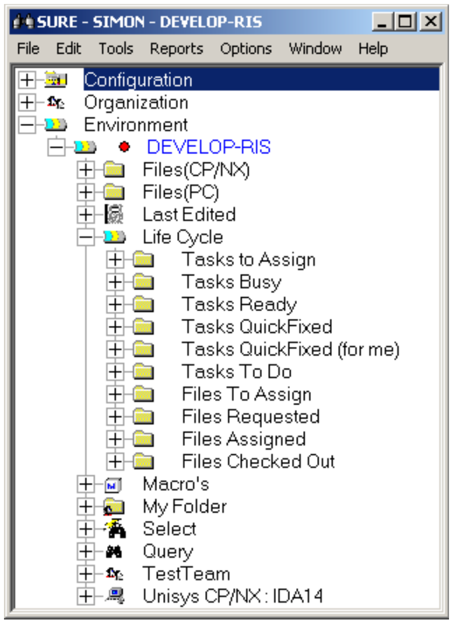
Environment	System	Project	Default	None	Global	SURE	Task
PRODUKTIE	DEMO						
PRODUKTIE							
DEVELOP							

- Task Names (SURE manual chapter ‘Task’)

  The name of your task, constructed by the SURE system at task entry, is by default derived from the project. The default name-standard is the project name separated by a dash and a four-digit number (e.g. PRJ-1234). It is possible to overrule this naming-standard by your own formula, or by a naming-standard for task-groups.

- Task Routing (SURE manual chapter ‘Task’)

  The task routing function affects the content of the workflow folders that are maintained by the system: Tasks To-Do, Tasks Busy, Tasks Ready, Tasks To-Assign and Tasks Quick-Fixed. The workflow queues are available for each environment and they contain tasks with a specific status (the ready-queue in the develop environment contains all tasks that have status DEVELOP and are marked as ready-(to-transfer)). These queues are visible in the user interface under folder Life-cycle:



  The SURE system performs the task routing based on a number of variables, one of them being the project for which the task is defined. The other variables that are evaluated are the employee function and the team:

  – A user can be assigned to a team and to one or more employee functions.

  – One or more project can be assigned to a team or a user.

  The tasks To-Do list contains the tasks that are assigned to your personal usercode or to one of your employee functions or to one of your teams, and that belong to a project which is linked to your team or your personal usercode.

  Using projects in combination with security can provide distinct separated logical parts in the repository. Note that the same functionality applies if you did not create projects, however the project name then is equal to the system name.

- Naming Standards (SURE manual chapter ‘Name Standards’):

  Naming standards are used by the SURE system for the construction of task names (see earlier in this section) and file-names.

  Naming standards for file names can be implemented as follows:

  – The new filename is determined by SURE according to a predefined naming standard formula.

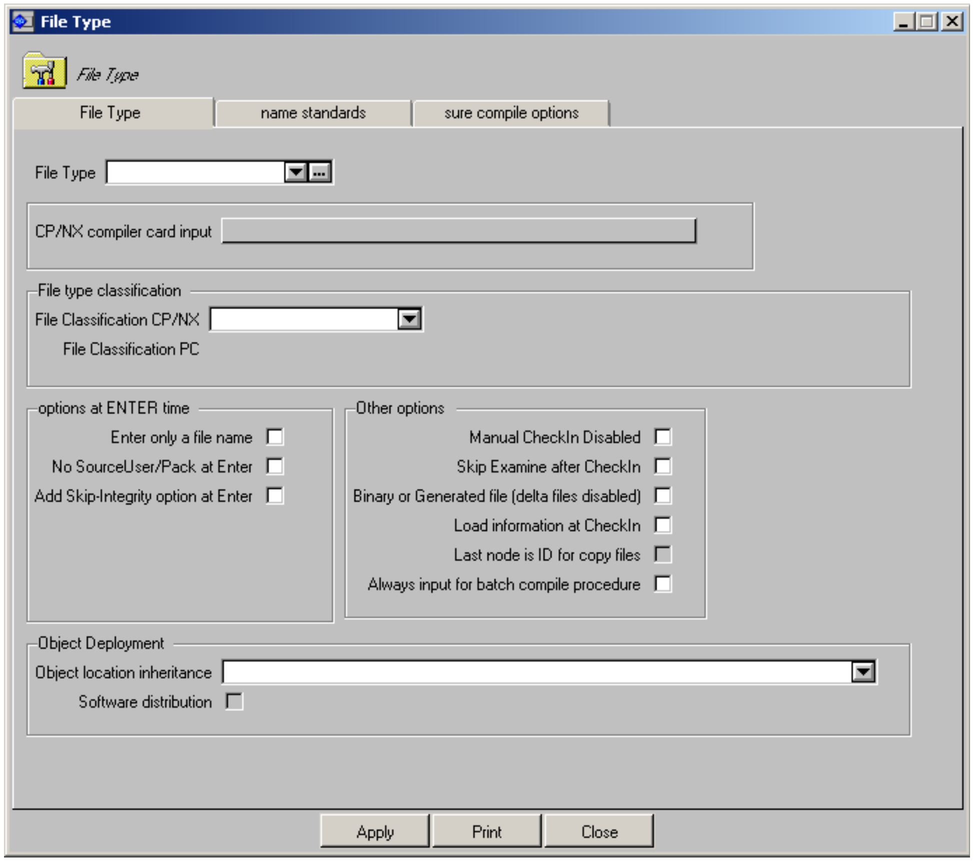
- The new filename is determined by the user, but checked by SURE according to a predefined naming standard formula.
- A mix of these two options.

The project name is one of the variables that you can use in this name standard definition.

The naming-standard formula must be defined per file-type.

It must be defined per system whether or not naming-standards have to be used.

10.6.4. (Configuration) File Type


File Type
File Type
File Type
name standards
sure compile options
File Type
CP/NX compiler card input
File type classification
File Classification CP/NX
File Classification PC
options at ENTER time
Enter only a file name
No SourceUser/Pack at Enter
Add Skip-Integrity option at Enter
Other options
Manual CheckIn Disabled
Skip Examine after CheckIn
Binary or Generated file (delta files disabled)
Load information at CheckIn
Last node is ID for copy files
Always input for batch compile procedure
Object Deployment
Object location inheritance
Software distribution
Apply
Print
Close

The file-type definition categorizes files with the same SURE characteristics. The file-types are separated for the CP/NX files and the PC files where both allow for slightly different characteristics. This document addresses the most common used file-type characteristics.

- CP/NX shared resources:

  These are copy-files (or other files) that must be visible for all team members in a shared work environment.

If you are using a shared work environment (see earlier in this chapter), then a file-type is required for copy files (or other files) that you want to be refreshed in that shared work environment after such a file is changed and checked-in to SURE.

The file-type option “Use as Copy file” in combination with the system option “Put copy files in the work location after Check-in/Load” will make SURE copy a file of this type to the work location. The work location contains a stable version of each copy-file. The developers use these stable copy-versions to compile their programs.

- CP/NX source files:

  Build support for the CP/NX files requires an object location to be defined for a file. This object location triggers the build support program (RESPECT/SURE/COMPILE) to invoke the appropriate compiler when the file must be compiled. The appropriate compiler is determined from the file-kind of the file. Source files are usually defined with a file-type with option “Inherit default object user/pack” set. This option triggers that the file inherits the object location defined for the system of the file. Therefore, the usercode, pack family name and hostname defined in the object environment for the system of the file, are applied to that file. As a result, the appropriate compiler is invoked for this source file resulting in an object file, which is copied to the destination by the distribution support.

  By default, the object name for a source file is named OBJECT/<source>. For individual files, you can alter this default under the file properties. If there is a company wide name standard for object names, then that name standard may be coded in a library, which is defined in the global options.

  Copy files usually do not have option “Inherit default object user/pack” set, and that results in skipping the copy-files for the actual compilation.

- CP/NX data files:

  If you have data files that are stored in the SURE repository, a number of options may be of interest.

  - First of all the SURE build support will never invoke a compiler, because the file kind does not match a supported compiler on the CP/NX system.
  - If you want these data files to be refreshed in the shared work location, you need to set the file type option “Use as Copy file” in combination with the system option “Put copy files in the Work environment after Check in/Load”.
  - If you want these files to be copied into your production environment by the distribution support, you need to set file type option “Inherit default object user/pack”.

  Note that you can check-in, checkout and transfer any type of file, also binary files. The binary files require two additional definitions, one which will skip the delta file creation and one that will skip the examine process. The corresponding options are Skip delta files after check in and Skip examine after check in. These two options are also applicable for CP/NX data files.

- PC input build files:

  PC input build files are files that are used as input in a build process that creates PC executables. Example of PC input build files are: make-files, command-files, source-files, header-files, project-files etc.

  These build files are the primary input files to the build process. This definition excludes the intermediate files and it also excludes the resulting binaries or executables.

The build process on the PC is responsible for downloading changed files to your PC and thereafter starting the build command for a file or a directory. A file must have its object-build- server defined to make this download happen.

The mechanism is similar to the object-inheritance mechanism for CP/NX files: The file-type option “Inherit default object user/pack” in combination with the system attributes for the object environment defines the inherited build server for a file.

With this object server defined for a file, the build process has its parameter for download and the build process.

Notice that the build server name is a virtual name; it does not have to be an actual server name. Furthermore, the download directories are defined in the PC SURE options, and they are usually different per environment. For this reason, the build server name may be equal for every environment.

- PC distributed files:

  PC distributed files are files that need to be distributed in your PC network. SURE can download a directory with all the files that need to be distributed, but SURE is not responsible for the actual distribution (putting the downloaded directory on a CD-rom and deploy, etc.)

  A PC distributed file may be a manually checked in file like an EXE, DLL, INI, BAT or any file, which can be part of the standard deploy process. On the other hand: it can also be an EXE, DLL, INT or any file that is the result of the build process controlled by SURE.

  A “manually checked-in distributed file” must have a file-type with option “software distribution” set.

  A “distributed file as a result of the build process” is automatically loaded by the SURE build process. A file resulting from a build process is slightly differently defined within the SURE repository. The first possibility is that you define a “Naming standard for the executable name at enter” with the “file type for loaded executable at enter” for a file type. This implies that if a file is loaded with this file type, an executable file is defined for this file. Secondly, you may define an executable file manually, via file properties → configuration → executables.

  Normally an executable file is a binary file, therefore it should not create delta files, should not make part of the examine process and it should be distributed. These settings can be achieved by the following options: “No delta files after check in”, “Skip examine after check in” and “Software distribution”.

- PC binary files (MSOffice):

  PC binary files are files for which no delta files are created. When no delta files are created, no difference and reconstruct facilities are present.

  A PC binary file is a manually checked in file, where SURE is used as archive mechanism. So Microsoft office files or any other file may be used in this mode.

  Example: a “manual checked-in distributed file” has the file type option “no delta files after check in” set, which results in ignoring delta files for every file of this type.

- PC copy files:

  PC copy files are files that are used as copy or include files within PC languages recognized by SURE. Current supported languages are C, C++ and Micro Focus COBOL. The SURE software scans sources of the previously mentioned languages for references to copy-files. Often only a part of the name is identified in the include statement, namely the last node of the file. The PC compiler finds the actual file using an environment variable like INCLUDE.

The examine process uses the file type option called ‘Last Node is used as ID for copy-files’ to achieve the same type of behavior. As a result of this option, the source file is connected to the actual full copy-file name if it exists for the node.

- PC C, C++ or COBOL source files:

  PC source files are files containing a language recognized by the SURE examine process. The examine process scans sources for references to copy-files and stores that information in the repository. Currently the languages C, C++ and Micro Focus COBOL are supported. Although any type of language can be loaded into the SURE, the examine process only recognizes the above-mentioned languages.

  If the examine process relates a source with a copy file, it will use this information in the compile process. If the copy file is changed, and therefore automatically compiled, the SURE system will implicitly recompile all the sources using this copy file. This side effect of the examine process is mostly used for the supported languages.

  Setting this option requires the files to be loaded with a file type containing the indication “Use as C/C++” or “Use as Micro Focus COBOL”.

- PC text files:

  A PC text file is any file containing plain text, of which you want to track the changes. Therefore, a PC text file may be an INI, BAT, PAS, TXT, and et cetera file. In principle, you could edit such a file with notepad. These files are stored in the SURE repository with their delta files. Using the delta files you can always trace the changes that are made to the source.

  Defining PC text files requires a file-type with the option “Skip examine at Check In” set.

The following table summarizes the described file types:

Type of File	Use as Copy File	Put Copy in Work	Inherit … Object User/ Pack	Skip Delta File	Skip Examine	Last Node Used as ID for Copy	Software Distri-bution	C, C++, COBOL
CP/NX shared resources	✓	✓						
CP/NX source files			✓					
CP/NX data files		?		✓	✓			
PC input build files	?		✓			?		?
PC distributed files				?			✓	
PC binary files (MS Office)				✓				
PC copy files	✓					✓		✓
PC C, C++ or Cobol source files	?	?	?					✓
PC text files							?	

✓ = This option is required for this file-type.

? = This option may be applicable for this file-type.

10.6.5. (Configuration) File

The file is the smallest entity stored in the repository. The file (and its contents) is stored in every environment in the repository.

Except for the file contents, a lot of other information is stored around the file entity and used within the SURE functionality:

- At Load or check-in time, the file content is stored in the repository. At checkout time, the file content is retrieved from the repository and made available through a standard file. Therefore SURE is required at check-in/out time. On the other hand: a developer does NOT need SURE while he is working on the source.

  In general, you can use any tool in the development environment, so development may still be done in the same way as always is done. This applies to CP/NX files and PC/UNIX files.

  For the CP/NX files SURE offers a partial workbench, which allows usage of various PC editors integrated with the compilers on the CP/NX machine. SURE allows local file editing in combination with a smooth integration with the CP/NX compilers. Thus, the file is edited on the PC and compiled on the CP/NX machine without any special actions from the developer.

  If you are using NX/Edit, which also contains a partial workbench, then you won't use the full SURE workbench. In that case SURE is only used for check-in/out but NX/Edit handles the compilation of the source.

- SURE contains a version indication, which consists of a major and a minor number separated by a point. The major number indicates how many times a file is transferred from one environment to another environment. The minor number indicates how many times the file was checked in to the environment. An example is given in the following table:

Environment	Step 1: Check Out/In	Step 2: Check Out/In	Step 3: Transfer	Step 4: Transfer	Step 5: Check Out/In
DEVELOP	5.1 -> 6.1	6.1 -> 6.2	6.2	6.2	6.2 -> 7.1
TESTING	5.1	5.1	6.2	6.2	6.2
PRODUCTION	5.1	5.1	5.1	6.2	6.2

  Step 1 and 5: The first check-out/in in a cycle raises the major number by one.

  Step 2: A second check-out/in in a cycle raises the minor version number by one.

- SURE does not allow checking out a file in the same environment by more than one user code at the same time (unless the baseline method is activated). However, it does allow checking out the file in different environments at the same time. Therefore, while the file is checked out in development, it can also be checked out in production. This mechanism allows solving production errors without disturbing the development process. An example is given in the following table:

Environment	Step 1: Check Out/In	Step 2: Check Out/In	Step 3: Reprocess QF	Step 4: Transfer	Step 5: Transfer
DEVELOP	5.1 -> 6.1	6.1	6.1 -> 6.2	6.2	6.2
TESTING	5.1	5.1	5.1	6.2	6.2
PRODUCTION	5.1	5.1 -> 5.2	5.2 (reprocessed)	5.2	6.2

Step 1: the source is changed in develop.

Step 2: the source is quick-fixed in production.

Step 3: the quick fix of production is reprocessed in develop.

Step 4 and 5: the develop sources (with the develop patch and the reprocessed quick fix) is transferred to acceptance and production.

- SURE keeps delta files of two successive file-versions in an environment. With these delta files, it is possible to rebuild any old version of an individual file.

Environment	Delta files between file version
DEVELOP	- 5.1 and 6.1- 6.1 and 6.2
TESTING	- 5.1 and 6.2
PRODUCTION	- 5.1 and 5.2- 5.2 and 6.2

- Source management is a main function for SURE, however other integrated functions such as Build support, Distribution support, Life-cycle support via task management also require definitions at file level. For this reason, the file entity contains a lot of additional information, which can be set via the file property function. A summary of information stored at file level is given in the next table

Build Support	Build Server	PC/UNIX
	Build Command	PC/UNIX
	Executable	PC/UNIX
	Task Attributes	CP/NX
	Binder specification	CP/NX
	Multiple Object Names	CP/NX
	Post Compile Processing	CP/NX
Distribution Support	Distributed file	PC/UNIX
	Object Location	CP/NX
Life-cycle Support	Current Task	
	Historical Task	
	Delta Files	
	Log	
	Run Time Statistics	CP/NX
Query Support	Copy/Include Files	
	Database	CP/NX
	Dataset	CP/NX
	etc.	CP/NX

- Files can be entered in the system in various ways. Initially files will be loaded per directory. For both file categories (CP/NX and PC), a batch load facility is present in SURE.
  - CP/NX files:

Select a directory or file from the CP/NX source directory and select function ‘Load in SURE’ from the property menu.

– PC Files:

  Select function ‘Tools→PC-Environment→Load files in SURE’ from the top-level menu. Files can be loaded from the defined local work directory in the SURE option.

Loaded files require the attributes ‘Project’ and ‘File Type’. SURE uses these attributes to complete the individual file definition.

- Files can also be entered in the system using the Edit/New/File function from the main menu. This function will normally be used when new files are created in an existing project. However, the load function is always available and can always be used to load a single file or a directory of files.

  The following table shows the functions that affect the life-cycle of a file:

Creation:	**Modification:**	**Deletion:**
Edit/New/File	Check Out/Check In	Delete
Load using initial load option	Recover	Purge
	Replace	Rename
	Reprocess Quick Fix	
	Apply Quick Fix	
	Load using replace option	

10.7. Configuration) Task Entities

- Task Priority
- Task Type
- Task Group
- Task

All definitions for tasks are assembled using the previously mentioned entities.

- A task priority categorizes a task in different priorities.
- A task type defines a task category including the start environment for the task. So normally you would create task types for each environment where you are going to make changes.
- A task group may combine a task type and a project. In this case, the name of the task is derived from the task group. If a task group is not used, the project name must be entered at task definition, and the name of the task is derived from the project name.

Chapter ‘Task’ of this manual gives detailed information about tasks, task-types and task-groups, and about the flow of a task in the EDP department.

10.7.1. (Configuration) Task

The task is the major controlling entity for life-cycle support, and defined at a global level in the repository. Global level means: the attributes of a task are visible and equal in all the environments. So if you select a task in the environment production and you show the properties, you will see exactly the same information as in develop or in any other environment.

This differs from the FILE concept, because a file is stored in the repository at environment level. The attributes of a file in environment production are not always equal to the attributes of a file in the develop environment.

The task is the functional unit of work. It contains various descriptive texts or an attachment. Changes to files are linked to a task by issuing the task name at check-out time. A task can be transferred to the next environment when all linked files are checked-in. If the task is transferred to the production environment, then the task-status becomes ‘solved’, and all linked files are added to the history of the task, and the task is added to the (logical) history of each linked file.

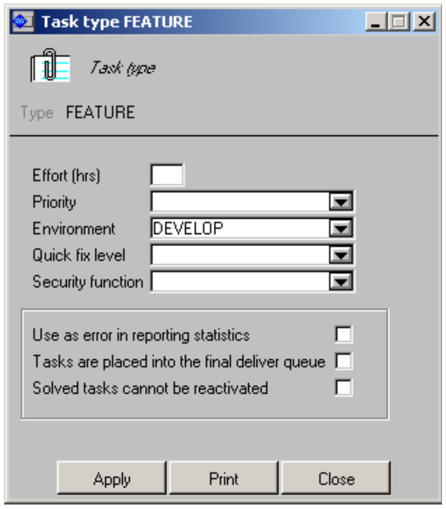
The task life-cycle can be integrated with an email system. If a task is assigned to a group of people or to an individual, an email can be sent to an email address. If a task changes from status, an email may be sent to the issuer of the task.

The task entry form can be customized so that it requires minimal input.

The following table defines the functions that affect the life-cycle of a task:

Creation	**Status Change**	**Close**
Edit/New/Task	Assign	Close
	Priority	Deny
	Ready	Delete
	Transfer	
	Activate	

10.7.2. (Configuration) Task Type



Type FEATURE

Effort (hrs)
Priority
Environment DEVELOP
Quick fix level
Security function

Use as error in reporting statistics ☐
Tasks are placed into the final deliver queue ☐
Solved tasks cannot be reactivated ☐

The 'task-type' is the only attribute for a task that has impact on the life-cycle functions that can be performed using the task. This is because the environment where the task starts (the task-environment) is inherited from the task-type. A newly added task inherits the task-environment from its task-type.

The task-environment determines the environment where fixes for tasks of this type must be made. You can only change a file if you are linked to a current task, and the changes must be made in the task-environment of that task.

The task-status determines if it is allowed to make changes for the task at this moment. You can only change a file if the status of your current task is equal to the task-environment of your current task.

Example:

- Suppose you have a task with environment DEVELOP and status DEVELOP. It is now possible to make this task current in the develop environment and to change a file because of this task in the develop environment.
- If you transfer the task to the acceptance environment, then the task-status becomes ACCEPTANCE. You cannot make changes in acceptance because the task-environment is still DEVELOP, and you cannot make changes in develop because the task-status is ACCEPTANCE.
- If you reactivate the task again in develop then the task-status becomes DEVELOP. You can now make additional changes in the develop environment.

A second important attribute of the task-type is the quick fix level.

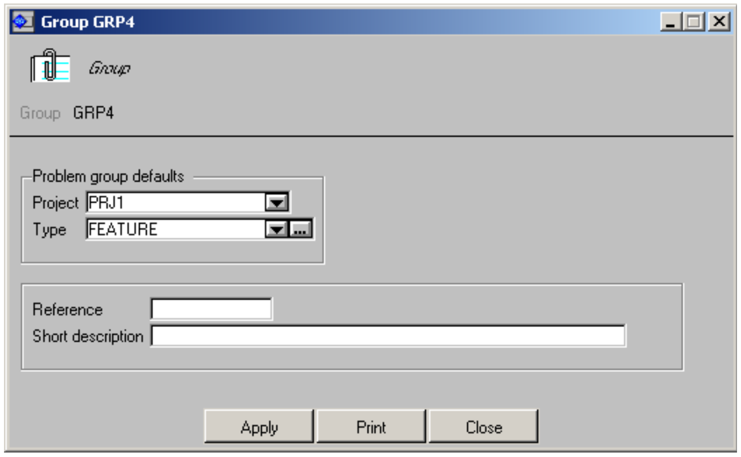
A quick fix is a modification to a file in an environment, which is not the lowest environment for that file. Example: consider a file that was created in the develop environment and transferred to acceptance and production. The lowest environment for this file is now the develop environment and a quick fix is a modification to the file in acceptance or in production.

The quick-fix-level identifies the environment where a task with this task-type can be quick-fixed. If this field is not entered, then the task cannot be used for quick fix purposes.

A separate task type is required for each environment where you want to make changes to files. For quick fixes, you can set the environment and quick fix level to the named environment. The following table shows the settings for the previous example.

Task-type FEATURE:	Environment	= DEVELOP
	Quick Fix level	= <empty>
Task-type QUICK-FIX:	Environment	= PRODUCTION
	Quick Fix level	= PRODUCTION

The files can be modified in develop for regular maintenance and in production for quick fixes.



Group GRP4

Problem group defaults
Project PRJ1
Type FEATURE

Reference
Short description

10.7.3. (Configuration) Task Group

A task group combines a task type with a project and additionally a short description.

Task groups are used for two reasons:

- The task naming facility.

  Task-names of newly added tasks are automatically determined by SURE. If you are using task groups then the new task name is based on the name of the task-group. If you are not using task groups, then the new task name is based on the project name.

- A shortcut when a new task is added:

  A project can be linked to a task-group.

  A task-type can be linked to a task-group.

A task-environment can be linked to a task-type.

A priority can be linked to a task-type.

Each task must have a task-type, a project and a task-environment. These three attributes are required if a new task is added to SURE. (Priority is an optional task attribute). A newly added task inherits the task-type and project from the task-group, and the task-environment from the task-type. If these three attributes are found via inheritance, then it is enough to enter the task-group when a new task is added; otherwise, the missing attributes have to be entered too.

A static task-group is a task-group that contains all three attributes: each task that is added via a static task group has a similar name (based on the name of the task group) and the same task-type, project and task-environment.

A dynamic task-group is a task-group that misses one (or more) of the three attributes: each task that is added via a dynamic task group has a similar name (based on the name of the task group), but the task-type, project or task-environment can differ.

10.7.4. (Configuration) Task Priority

The task priority is used for query and reporting purposes.

The customer can define his own list of task priorities via explorer folder Configuration/Drop Down box value/Task Priority.

It is possible to define special customized workflow folders via the ‘Macro’ function. The task-priority can be used in the task-selection-expression of such a customized workflow folder.

The task popup menu allows changing of this priority field. For this reason, the priority field can be used for moving tasks from one customized workflow folder to another.

10.8. (Configuration) Organization Entities

- Employee Function
- Team
- User

For a user there are two important entities:

- The employee function or the user’s role in the organization. The employee function contains a security map, which defines the actions that may be performed. Secondly, an employee function is used for task routing purposes. A task can be assigned to an employee-function and then all users with that employee-function receive the task in their To-Do lists.
- A team is a group of people teamed together as a work unit. A team can also be used for task routing purposes. A task can be assigned to a team or to an employee function. Together with the assigned projects for a user, the workflow queues are assembled.

The following paragraphs describe the above-mentioned entities with more detail.

Chapter ‘Authorization Mechanism’ of the manual gives detailed information about usercodes, employee functions, teams, and the authorization mechanism.

Chapter ‘Task’ of the manual gives detailed information about To-Do lists.

10.8.1. (Configuration) Employee Function

The employee function is a role within a company performed by a group of employees. A usercode can be linked to zero, one or more employee functions. For this reason, one should choose elementary roles within a company.

The employee function is used in two different ways:

- Security:

  Every employee function may contain a security map. A user is linked to one or more employee functions. The combined security maps of these different employee functions determine the function matrix of the user.

- Task Routing:

  A task can be assigned to an employee function. Every user who has the employee function gets that task in his To-Do list.

  If a user is assigned to one or more projects then only the files/tasks of those projects are visible for him. For example: tasks of other projects won’t appear in his To-Do list, even if these tasks are routed to one of his employee-functions.

10.8.2. (Configuration) Team

A team is a group of usercodes. A usercode can be linked to zero, one or more teams. Each user of a team may have different employee functions.

The team definition is used for the following:

- Project assignment:

  A team can be linked to zero, one or more projects. A user inherits the (merged) project-lists of his teams. The merged team-project-list overrules an individual project list of the user.

  The project-list works as a filter: if user does not have a project-list (directly or indirectly via his teams) then he can see all files. If he has a project-list then he can only see the files of his projects.

- Task Routing:

  A task can be assigned to a team. Each member of the team gets this task in his To-Do list. The security is still defined at employee function.

- Contents of Life-cycle Work Flow folders:

The content of the various team life-cycle folders is affected by the team definition. For example, the folder Life-cycle/Team Task/Busy contains all the busy tasks of the members of all my teams.

10.8.3. (Configuration) User

The user entity contains an entry for each user that can log-on to the system. A usercode that is not announced in SURE cannot log-on to SURE. The log-on procedure validates the usercode and password against the userdatafile. For this reason, each SURE user also needs to be defined in the userdatafile.

The user definition supports (amongst others) the following:

- Project Definition:

  A user can be linked to zero, one or more projects. If a user is NOT linked to any project then all files/tasks of all projects are visible for him. If a user is linked to one or more projects, then only the files/tasks of those projects are visible for him.

- Team Definition:

  A user can be linked to zero, one or more teams. These teams determine the project assignment of the user and affect the contents of his life-cycle workflow folders.

  A project-list that is inherited from the team(s) overrules an individual project-list.

- Employee Function:

  A user can be linked to zero, one or more employee functions. These employee functions determine the security map for a user and affect the contents of his To-Do list.

- Individual Security:

  Security may be defined at individual user level. The usercode where the SURE installation is started contains an individual security map. With this installation user the rest of the system can be configured. An individual security map overrules the security map that was inherited from the employee functions.

- Default environment:

  A user is linked to a SURE-environment. Each time when he logs on to SURE, he will be routed to his default environment. From there, he can navigate to other environments.