SG24-4610-00

International Technical Support Organization

TeamConnection and WorkFrame Integration Survival Guide

July 1996

IBM
International Technical Support Organization
San Jose Center

SG24-4610-00

International Technical Support Organization

TeamConnection and WorkFrame Integration Survival Guide

July 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xi.

First Edition (July 1996)

This edition applies to Version 1.0 of TeamConnection, Product Number 31H3744 and Version 3.0 of IBM VisualAge for C++ for OS/2, Program Number 5875-XXX for use with the OS/2 Warp Operating System.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471 Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Abstract

An integrated part of IBM VisualAge for C++ for OS/2 and IBM VisualAge for COBOL for OS/2, the WorkFrame environment has become the environment of choice for developing applications with these and other 3GL languages. Together with TeamConnection, IBM′s state-of-the-art library and repository, the WorkFrame environment forms a very strong base for the development of 3GL applications. This redbook shows how to integrate TeamConnection with the WorkFrame environment without losing the benefits of WorkFrame′s tightly integrated and project-oriented compile, test, and debug. The book also shows how to take advantage of the integrated build facility of TeamConnection and set up a new project by using the REXX programs described in the book.

This redbook is written for system administrators, project leaders, developers, and anyone else who has an interest in learning how to integrate IBM VisualAge for C++ or IBM VisualAge for COBOL WorkFrame with TeamConnection. The book includes examples of how to set up a new or an existing project by using WorkFrame and TeamConnection together. The examples and *time limited* versions of TeamConnection and VisualAge for C++ are also included on a complementary CD-ROM.

(210 pages)

Contents

Figures

Special Notices

This publication is intended to help system administrators, project leaders, developers, and others who want to integrate IBM VisualAge for C++ or IBM VisualAge for COBOL WorkFrame with TeamConnection. The information in this publication is not intended as the specification of any programming interfaces that are provided by TeamConnection or the WorkFrame products. See the PUBLICATIONS section of the IBM Programming Announcement for TeamConnection, IBM VisualAge C++ for OS/2, and IBM VisualAge for COBOL for OS/2 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Preface

This redbook is intended for system administrators, project leaders, developers, and anyone else who has an interest in learning how to integrate IBM VisualAge for C++ or IBM VisualAge for COBOL WorkFrame with TeamConnection. The book shows the user how to integrate TeamConnection with the WorkFrame environment without losing the benefits of WorkFrame′s tightly integrated and project-oriented compile, test, and debug. The book also shows how to take advantage of the integrated build facility of TeamConnection and set up a project by using the REXX programs described in the book.

This redbook and the accompanying REXX examples were written during a residency at the ITSO San Jose Center (Almaden Research Center, San Jose, California) during February and March 1996. The goal of the residency was to look at and, if possible, improve the current integration between TeamConnection and WorkFrame. Thus the book takes you through the phases of setting up TeamConnection and WorkFrame and discusses such topics as naming standards, project structure, problem tracking, and build considerations.

The book comes with "ready-to-run" examples that you can modify to fit your own needs. The examples and *time limited* versions of TeamConnection and VisualAge for C++ are also included on a complementary CD-ROM. We hope that you find this redbook valuable in your effort to integrate TeamConnection and WorkFrame.

Enjoy reading.

[illegible]

Leif Trulsson
ITSO - San Jose, California
July 1996

How This Redbook Is Organized

The redbook is organized as follows:

- Chapter 1, “Introduction”

  This chapter gives a short introduction to the book.

- Chapter 2, “Our Project Development Environment”

  This chapter describes our project and what we expect from TeamConnection and its integration with WorkFrame.

- Chapter 3, “Setting Up TeamConnection”

  This chapter provides information on how to set up TeamConnection and discusses the various issues to consider.

- Chapter 4, “Using TeamConnection and WorkFrame”

  This chapter describes the use of TeamConnection and WorkFrame together.

- Chapter 5, “Setting Up WorkFrame”

  This chapter describes how to set up WorkFrame with the supplied project template. It also describes how to customize the project and the defined project actions.

- Chapter 6, “Using WorkFrame Projects”

  This chapter describes how to migrate existing WorkFrame projects to TeamConnection.

- Appendix A, “Sample REXX Programs”

  This chapter describes all of the REXX programs we used in our integration effort.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

TeamConnection

- *IBM TeamConnection for OS/2 Getting Started*, SC34-4498
- *IBM TeamConnection for OS/2 User's Guide*, SC34-4499
- *IBM TeamConnection for OS/2 Commands Reference*, SC34-4501
- *IBM TeamConnection for OS/2 Messages*, SC34-4502

VisualAge for C++

- *Welcome to VisualAge for C++*, S25H-6957
- *VisualAge for C++ User's Guide*, S25H-6961

VisualAge for COBOL

- *IBM VisualAge for COBOL for OS/2 WorkFrame User's Guide*, SG24-4604

IBM OS/2 Publications

- *REXX User's Guide*, S10G-6269
- *REXX Reference*, S10G-6268

International Technical Support Organization Publications

- *Introduction to the IBM Application Development Team Suite*, SG24-4648
- *Did You Say CMVC?*, GG24-4178
- *Family Planning and Application Development - TeamConnection Unleashed*, SG26-2008

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Most major credit cards are accepted. Outside the USA, customers should contact their local IBM office. For guidance on ordering, send a note to BOOKSHOP at DKIBMVM1 or E-mail to bookshop@dk.ibm.com.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser to the following URL:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. The internal Redbooks home page may be found at the following URL:

```
http://w3.itso.ibm.com/redbooks/redbooks.html
```

Subscribing to Internet Listserver

IBM redbook titles/abstracts are now available through Internet E-mail via the IBM Announcement Listserver. With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. All it takes is a few minutes to set up a profile, and you can get news (in ASCII format) from selected categories.

To initiate the service, send an E-mail note to:

```
announce@webster.ibmlink.ibm.com
```

with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

To obtain more details about this service, employees may type the following:

```
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
```

Note: INEWS users can select RelInfo from the action bar to execute this command automatically.

Acknowledgments

This project was designed and managed by:

Leif Trulsson
International Technical Support Organization, San Jose Center

The authors of this redbook are:

Lutz Sparmann
IBM Germany

Yuhsuke Watanabe
IBM Japan

Leif Trulsson
International Technical Support Organization, San Jose Center

This publication is the result of a residency conducted at the International Technical Support Organization, San Jose Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this redbook:

Maggie Cutler, Editor
International Technical Support Organization, San Jose Center

Stephanie Manning
International Technical Support Organization, San Jose Center

The project leader also would like to thank the following people for their invaluable assistance in making it all happen:

Elsa Barron
International Technical Support Organization, San Jose Center

Mary Comianos
International Technical Support Organization, San Jose Center

Alan Tippett
International Technical Support Organization, San Jose Center

Chapter 1. Introduction

Combining TeamConnection and WorkFrame will give you a well-controlled, user-friendly, and flexible environment for your application development.

WorkFrame′s flexibility and its ability to integrate nearly any kind of tool (such as compilers, editors, browsers, and debuggers) into a versatile development environment and TeamConnection′s integrated build, packaging, and strength in software configuration management and version control make it worth your while trying to integrate them in a way that enables you to take full advantage of both products.

In this book we show you one way of integrating TeamConnection and WorkFrame. You can use it as is or modify it to suit your own needs and what works best for your environment.

The book is based on TeamConnection Version 1.0 fix level 2 and VisualAge for C++ Version 3.0, WorkFrame fix level CTW302.

Chapter 2. Our Project Development Environment

In this chapter we talk about our *fictive* project and the development environment. We assume that the reader is familiar with the basic concepts of TeamConnection and WorkFrame.

The Project

We, **ABC Corporation**, have decided to use an OS/2-based development environment consisting of TeamConnection and WorkFrame to develop a new client/server project or product called **FirstShot**.

FirstShot will consist of a:

- **Server building block** that can work with DB/2 and Oracle database systems
- **Client building block** providing a command line interface as well as a graphical user interface (GUI)
- **GUI building block**

Each of these building blocks will finally be available for several target platforms: **OS/2**, **AIX**, and **SUN**.

Figure 1 shows the functional structure of **FirstShot**.

```
                                             ┌---Common Code
                      ┌--Database Interface--┼---DB/2 Specifics
                      │                      └---Oracle Specifics
                      │
           ┌--Server--┼--All other server functions
           │ Building │
           │  Block   │                           ┌---Common code
           │          └--Communications Interface-┼---NetBIOS
           │                   (to clients)       └---TCP/IP
           │
           │                                      ┌---Common Code
           │          ┌--Communications Interface-┼---NetBIOS
           │          │        (to server)        └---TCP/IP
           │          │
FirstShot--┼--Client--┼--All the other client functions
           │ Building │
           │  Block   └--GUI Support Interface
           │
           │          ┌-----GUI-Client Interface
           │          │
           └--GUI-----┼-----All other GUI functions
            Building  │
             Block    │                            ┌---Common Code
                      └-----Presentation Interface-┼---OS/2 PM
                                                   └---OSF/Motif
```

Figure 1. Functional Structure of FirstShot

Our library and build processes have to be able to control several “flavors” of the same executable at the same time, because we plan to keep one version of an executable for:

- Integration testing
- Source level debug

Because **ABC Corporation** does not have any experience with OSF/Motif and Oracle, we will use the service of two external subcontractors, EasySoft and QuickSoft, to get the coding done.

Development Environment Wish List

To end up with a family setup that best fits our needs, it is a good idea to first formulate our expectations and requirements. For example, think about the things you would like to query the library system about in the future! Knowing such queries will help you come up with a naming convention that might simplify (and speed up) database queries.

What We Expect from TeamConnection

We expect to be able to:

- Maintain more than one product in one library
- Allow controlled sharing of code among different products
- Have easy-to-find common functions
- Enable the same user groups to access different projects or products in the family
- Show the functional structure of the product
- List all parts belonging to a functional unit
- Keep design specification and product documentation with the product structure
- Assign defects and features to functional units
- Have common and platform-specific source files that have the same name
- Build code for different target environments, using the same file names
- Build different “flavors” of the same executable for the same target platform:
  - Code for function test and integration (built using standard compiler options)
  - Source level debug version of code (built using debug-specific compiler options that can generate additional output files)
- Identify or list all subcontractors
- Restrict access of subcontractors
- List all user IDs of a specific subcontractor
- Identify or list all components a single user or group of users can access
- Find out who has access to which components

What We Expect of the Integration

We expect the following of the integration:

- One user can work on multiple work areas or different projects or products at the same time on the same PC.
- Library functions can be executed on WorkFrame parts and projects.
- A tight edit-compile-debug loop will be supported; that is,
  - Compiler error messages are displayed and selectable in a scrollable window.
  - Double-clicking on a compiler error message jumps to line in the editor.
  - Switching to code generation for source level debug will be easy.
- The latest code version is always in the library.
- A build should always be done with the library build setup.
- WorkFrame projects can automatically be generated based on a TeamConnection work area. (The WorkFrame project has to include the WorkFrame tools setup.)
- A WorkFrame project can be automatically imported to a TeamConnection work area.

Chapter 3. Setting Up TeamConnection

In this chapter, we show you how to set up TeamConnection. We also take a look at the following issues:

- Basic concepts and restrictions
- Family administration issues such as:
  - Name spaces
  - User groups
- Grouping the parts and controlling access
- Structural considerations such as:
  - Project or product structure
  - Problem tracking
  - Common functions
- Naming the parts
- Build considerations such as:
  - Keeping versions of parsers and builders
  - Using NULL builders

Basic Concepts and Restrictions

Before we start defining our TeamConnection setup, it is a good idea to review some basic concepts and restrictions of TeamConnection:

- **TeamConnection names**
  - Are all case sensitive!

- **Components**
  - Access and notification (inheritance) are controlled through components.
  - Defects and features are controlled by components.
  - Parts are grouped by components.
  - Component names must be unique per family.
  - Component names must not be longer than 31 characters.
  - Components can have one or more parent components.

- **Releases**
  - A release collects all parts making up a version of a product.
  - Release names must be unique per family.
  - Release names (currently) cannot be changed.
  - The length of a release name is limited to 15 characters.
  - Part names have to be unique per release.

- **Parts**
  - A part can be attached to only one component.
  - A part name consists of a base name and a path name.
  - A part name must be unique per component.
  - A part name must be unique per release.
  - You can define up to 20 configurable fields for parts.

- **Users**
  - The maximum length of the user name field is 31 characters.
  - The maximum length of the user area field is 15 characters.
  - You can define up to 20 configurable fields for users.

- **Defects and Features**
  - The maximum length of a defect or feature name is 15 characters.
  - You can define up to 20 configurable fields for defects and features.

GUI Capabilities

The GUI of TeamConnection with its *tree view* of the component structure is very useful for providing structural information about a project or product or navigating through the family.

Keep in mind, however, that the GUI is relatively slow and sometimes not well suited for the tasks a developer or an administrator has to perform during their daily work. So, when making a decision about how to organize a TeamConnection family, always consider the implications of the decision from both a command line and GUI point of view.

Naming Conventions

Because a component name has to be unique within one family, and to simplify the retrieval of information from the TeamConnection database, you first have to think about naming schemes.

The family administrator has to establish naming conventions that will be observed *familywide*. It will also be useful for the project or product owners to think about a *project* or *productwide* naming convention for both component and part names.

We discuss and introduce a naming convention in a step-by-step manner when discussing the different aspects of the TeamConnection setup for the project (see “Name Spaces” on page 10, “User Groups” on page 12, and “Naming the Parts” on page 25).

Family Administration Issues

In the sections that follow, we look at two family administration issues:

- Name spaces
- User groups

Name Spaces

Consider the following wish list item:

We expect to be able to:

- Maintain more than one product in one library

Because the component names have to be unique in the family, we have to define a *familywide* naming convention to generate separate *component name spaces* for the different projects.

In our family, a name space is defined by using a **three-character prefix** in every component name.

If the same person has to work with different projects at the same time (think about administration and problem reporting), it makes sense to introduce some additional familywide conventions:

- The root component of a project is always named **prd.root**, where prd is the prefix defining the project name space.
- All other components of the project are *children* of this *project root*.

We define the name space for **FirstShot** to be **fst**, and, because you always will need a place where you can attach and organize information related to family administration, we define the name space **adm** for this purpose (see Figure 2 on page 11).

Additional familywide conventions for component names will be introduced later when discussing the different aspects of the TeamConnection setup (see “User Groups” on page 12, “Grouping the Parts and Controlling Access” on page 16, “Project or Product Structure” on page 18, and “Naming the Parts” on page 25).

```
        ┌-adm.root
        │
root --┤
        │
        └-fst.root
```

Figure 2. The TeamConnection Component Structure (Iteration 1)

User Groups

Consider the following wish list items:

We expect to be able to:

- Enable the same user groups to access different projects or products in the family
- Identify or list all subcontractors
- List all user IDs of a specific subcontractor
- Find out who has access to which components
- Identify or list all components a single user or group of users can access

Within TeamConnection you can gain access to a component (and the parts belonging to it) by:

- Owning the component
- Adding a user to the access list of the component
  or
- Inheriting the access from parent components

TeamConnection enables you to group users by:

- Defining a component and adding the users to the access list
- Implementing a naming convention for the *user area*
  or
- Creating a configurable field for the user objects and using it for this purpose.

In an environment where the same groups of users have the same type of authority on files and components in different projects or products, it might soon become a significant effort to maintain separate access lists for all the projects or products and keep them synchronized.

For our family we decide to introduce an administrative component, **adm.user-groups** (see Figure 3 on page 14), that will be the parent of all the components used to group users and their authorities. The names of the user groups will be prefixed by **adm.ug** followed by the name of this user group. A user group name consists of an authority specification followed by the group identifier. We will use the following authority specifications:

dev	developers
bld	builders
tld	team leaders

To simplify the search for specific user groups such as subcontractors or temporary employees, we will start the group identifiers for these user group names with:

s-	for subcontractors

t-	for temporary employees

Examples:

`adm.ug.dev.s-easysoft`, `adm.ug.dev.s-quicksoft`, `adm.ug.dev.all`, `adm.ug.bld.all`, `adm.ug.dev.t-students`

We further decide to:

- Use the **user area** field to indicate the department to which the user belongs
- Prefix the user IDs of specific user groups to simplify the task of granting access to them:

s-	for subcontractor user IDs
t-	for user IDs of temporary employees

Examples:

`s-miller`, `t-frank.`, `turner`, `angela`

```
        ┌--adm.root
        │     │
        │     └---adm.user-groups
        │            │
        │            ├---adm.ug.dev.all
        │            │         ┌-------┐
        │            │         │mike   │
        │            │         │george │
        │            │         │ ...   │
        │            │         └-------┘
        │            ├---adm.ug.dev.t-temporary
        │            │         ┌---------┐
        │            │         │t-rmiller│
        │            │         │ ...     │
        │            │         └---------┘
        │            ├---adm.ug.dev.s-quicksoft
        │            │         ┌---------┐
        │            │         │s-miller │
        │            │         │ ...     │
  root--┤            │         └---------┘
        │            ├---adm.ug.dev.s-easysoft
        │            │         ┌---------┐
        │            │         │s-frank  │
        │            │         └---------┘
        │            │
        │            ├---adm.ug.bld.all
        │            │         ┌---------┐
        │            │         │peter    │
        │            │         │mike     │
        │            │         │george   │
        │            │         │ ...     │
        │            │         └---------┘
        │            │
        │            └---adm.ug.tld.all
        │                      ┌-------┐
        │                      │mike   │
        │                      │elsa   │
        │                      │ ...   │
        │                      └-------┘
        └--fst.root ...
```

Figure 3. The TeamConnection Component Structure (Iteration 2)

Note: TeamConnection delivers some sample REXX scripts, which can be used to display information about access rights and access inheritance:

- ACCCOMP componentname
  shows information about access rights
- ACCINHER componentname userlogin
  shows information about access inheritance

Grouping the Parts and Controlling Access

Consider the following wish list item:

We expect to be able to:

- Restrict access of subcontractors

TeamConnection components are designed to control access to parts. Because a part can be associated with only one component, we should group our parts so that access and notification management can be easily handled.

The only reasons not to attach all parts of a project or product to one single component are:

- To restrict the access to specific parts
- To establish a notification service for part changes
- To distinguish specific part categories

Because the problem of organizing the parts will be the same for all the projects or products in our family, we decide, for ease of use, to add a new item to the familywide naming convention for projects or products:

- Each project or product will have a component named **prd.source-parts** that will be the base for all components holding source code.

- Each source code component will start with the prefix **prd.src.** followed by the name of the source code group.

  Examples:
  `fst.src.anything-else`, `fst.src.quicksoft`, `fst.src.easysoft`

To be able to selectively handle object parts (that is, everything generated by a builder), we also decide that:

- Each project or product will have a component named **prd.object-parts** that will be the base for all components holding object parts.

- Each component holding object parts will start with the prefix **prd.obj.** followed by the name of the object parts group.

  Examples:
  `fst.obj.anything-else`, `fst.obj.subcontractors`

The components holding the parts are connected not only to the **prd.source-parts** or **prd.object-parts** trees but also to the **adm.user-groups** tree that defines the access rights of the different user groups to the parts (see Figure 4 on page 17).

Note: We do not try to group the source parts according to the functional structure of the project or product. The need to provide additional components for access restriction and notification purposes would tend to make such a structure confusing and unmanageable.

```
   ┌---adm.ug.dev.s-quicksoft -----------┐
   ├---adm.ug.dev.s-easysoft --------┐   │
   │                                 │   │
   ├---adm.ug.dev.all ----┐          │   │
 ┌-┤                      │          │   │
 │ ├---adm.ug.bld.all --┐ │          │   │
 │ │                    │ │          │   │
 │ └---adm.ug.tld.all   │ │          │   │
 │                      │ │          │   │
root -┐                 │ │          │   │
      │                 │ │          │   │
      ├---fst.root      │ │          │   │
      │      │          v │          │   │
      │      ├---fst.object-parts    │   │
      │      │            │          │   v
      │      │            │       ┌---fst.src.quicksoft
      │      │            v       │  │
      │      └---fst.source-parts-┤  v
      │                           └---fst.src.easysoft
      │
      │
      └---com.root ...
```

Figure 4. The TeamConnection Component Structure (Iteration 3)

Structural Considerations

In this section we talk about how to organize our component structure according to:

- Project or product structure
- Problem tracking considerations
- Common functions

Project or Product Structure

Consider the following wish list items:

We expect to be able to:

- Show the functional structure of the product
- Keep design specification and product documentation with the product structure
- List all parts belonging to a functional unit

The wish to document the functional structure of the project or product in TeamConnection as well as the desire to organize the design specification, the project documentation, and the parts by function imply that we will have to create a *functional component tree* for our project.

Since this idea also is very likely to be used by other projects, we decide to introduce a new familywide convention:

- If a project will contain a component tree that is a functional description of the system, the root of this tree will be named **prd.functional-view**.
- Each component in this tree will start with the prefix **prd.fnc.** followed by the name of the component.

  Examples:
  `fst.fnc.server`, `fst.fnc.client`, `fst.fnc.gui`

Our decision to have a functional organization (see Figure 5 on page 19) makes the TeamConnection component structure very much resemble the structure of **Firstshot** (see Figure 1 on page 4).

```
                                  root
                                   │
     ┌─────────────────────────────┴──────────────────────────────┐
     │                                                              │
 fst.root                                                       adm.root
     │                                                              │
     ├---fst.source-parts <------------- adm.ug.dev.all --┬--...--┘
     │   ...                                         ...    │
     └---fst.functional-view <---------- adm.ug.tld.all --┘
            │
            ├---fst.fnc.server
            │      │
            │      ├---fst.fnc.database-iface
            │      │      ├---fst.fnc.db-common-functions
            │      │      ├---fst.fnc.db2-specifics
            │      │      └---fst.fnc.oracle-specifics
            │      │   ...
            │      └---fst.fnc.srv-xfer-iface
            │             ├---fst.fnc.srv-xfr-common
            │             ├---fst.fnc.netbios (!)
            │             └---fst.fnc.tcpip   (!)
            │
            ├---fst.fnc.client
            │      │
            │      ├---fst.fnc.client-xfer-iface
            │      │      ├---fst.fnc.client-xfer-common
            │      │      ├---fst.fnc.netbios (!)
            │      │      └---fst.fnc.tcpip   (!)
            │      │   ...
            │      └---fst.fnc.gui-support
            │
            └---fst.fnc.gui
                   │
                   ├---fst.fnc.gui-client-iface
                   │   ...
                   └---fst.fnc.presentation-iface
                          ├---fst.fnc.gui-pres-common
                          ├---fst.fnc.os2pm-functions
                          │      └--- ...
                          └---fst.fnc.osf-motif-functions
                                 └--- ...
```

Figure 5. The TeamConnection Component Structure (Iteration 4)

Note: For **FirstShot**, we will maintain only one functional structure. Be aware, however, that it might be necessary (for access control reasons) to maintain more than one such functional structure, which may show the product structure in more or less detail depending on the purpose for which they are used (for example, design documentation, problem tracking).

Problem Tracking

Consider the following wish list item:

We expect to be able to:

- Assign defects and features to functional units

In TeamConnection every defect or feature is attached to a component. The feature *originator* is the user who opens a feature. The feature *owner* is, by default, the owner of the component to which the feature is assigned. Any user within the family can open a feature against any component in the hierarchy.

If you plan to use the TeamConnection problem tracking process, the way in which you handle the problem assignment might have an impact on your component structure. Basically you can handle problem assignment in two ways:

- Use a central component (for example, the project or product root) to which to attach all the defects.
- Assign a defect directly to the "right" person (component).

You can of course combine the methods.

Using one central component to which to attach defects might be a good solution if your product is in maintenance mode. The owner of the component is responsible for assigning the defect to the person in charge of handling it.

Assigning a defect directly to the right person (component) assumes that:

- You have a functional component structure.
- The person who describes a problem has a good knowledge about which function is affected by the problem and who may be in charge of solving it.

This approach is usually a good solution if a project or product is in the test or integration phase, where a large number of problems will be found and the people who find the problems are familiar with the product and its structure.

Using a functional structure to attach features or defects to is usually also better when you are using complex development processes that involve approval, sizing, and notification.

Function Tree versus Configurable Field

An alternative approach to describing the problem area by attaching a defect to a functional component is the use of a configurable field for the defect objects that could be called Area_Code. This field will have a set of predefined values that can be used to identify the function (and person) to which a problem belongs.

Together with a central component for defect assignment, this approach might offer a faster method of creating a defect—rather than having to navigate through the functional component tree for every new defect.

Another advantage of this approach is that your problem area codes need not match the functional structure.

A slight disadvantage of this approach is that a defect always will have to be reassigned before the person responsible for handling it can start working on the defect. As this is probably the most common way of dealing with the assignments of defects and features, we consider this a very minor disadvantage.

Note: For **FirstShot**, we will maintain only one functional structure. Be aware, however, that it might be necessary (for access control reasons) to maintain a separate functional structure for defect and feature handling.

Because the function tree approach is very likely to be used by other projects or products, we decide to introduce a new familywide convention:

- If a project or product will contain a separate functional component tree for defect and feature management, the root of this tree will be named **prd.defects-and-features**.
- Each component of this tree will start with the prefix **prd.d&f.**, followed by the name of the component. If there are multiple functional trees, it is a good idea to use the same names in the different trees (see Figure 6 on page 22).

  Examples:

  `fst.d&f.server`, `fst.d&f.client`, `fst.d&f.gui`

```
         ┌---adm.root
         │
         │             ┌---fst.defects-and-features
         │             │      │
         │             │      ├---fst.d&f.server ...
         │             │      ├---fst.d&f.client ...
         │             │      └---fst.d&f.gui ...
         │             │
         │             ├---fst.source-parts
root --┤---fst.root -┤
         │             ├---fst.object-parts
         │             │
         │             └---fst.functional-view
         │                    │
         │                    ├---fst.fnc.server ...
         │                    ├---fst.fnc.client ...
         │                    └---fst.fnc.gui ...
         │
         └---...
```

Figure 6. The TeamConnection Component Structure (Iteration 5)

Common Functions

Consider the following wish list items:

We expect to be able to:

- Allow controlled sharing of code among different products
- Have easy-to-find common functions

If you are thinking about larger families with multiple projects or products sharing code, you might have to reconsider the placement of parts that are used by more than one project or product. The basic reasons for this are:

- Changes to a *common function* involve many different developers as compared to when a function is used in one project or product only. Having a separate component structure, using a more complex release process, and having specific notification lists will turn out to be useful in controlling *common functions.*

- Having one central place to look would significantly simplify the search for an existing common function.

- A component structure that groups common functions in any suitable way in TeamConnection would offer a list of the existing common functions that is always up-to-date.

A collection of common functions is nothing other than another project or product, so we create a new name space (**com**) for this purpose.

The root component for the common functions will be **com.root**. The categorization and grouping of the common functions will be done in the component tree, starting at **com.functional-view**. This functional component tree would also be the right place to attach notification lists, defect reports, and requests for new features. The leaves of this component tree will identify the different common functions available (see Figure 7 on page 24).

Note: For the common functions it might be appropriate to attach the sources as well as the access and notification lists directly to the functional component tree.

```
┌--fst.root
│     │
│     ├---fst.source-parts
│     │  ...
│     └---fst.functional-view
│            │
│            ├---fst.fnc.server
│            │      │
│            │      ├---fst.fnc.database-iface
│            │      │      ├---fst.fnc.db-common-functions
│            │      │      ├---fst.fnc.db2-specifics
│            │      │      └---fst.fnc.oracle-specifics
│            │      │  ...
│            │      └---fst.fnc.srv-xfer-iface
│            │             └---fst.fnc.srv-xfr-common
│            │
│            ├---fst.fnc.client
│            │      │
│            │      ├---fst.fnc.client-xfer-iface
│            │      │      └---fst.fnc.client-xfer-common
│            │      │  ...
│            │      └---fst.fnc.gui-support
│            │
│            └---fst.fnc.gui
│                   │
│                   ├---fst.fnc.gui-client-iface
│                   │  ...
│                   └---fst.fnc.presentation-iface
├--com.root
│     │
│     └---com.functional-view
│            │
│            ├---com.fnc.xfer-functions  ┌-----┐
│            │      │                    │jean │
│            │      │                    │mike │
│            │      │                    └-----┘
│            │      │
│            │      ├---com.fnc.netbios
│            │      └---com.fnc.tcpip
│            │
│            └---com.fnc.gui-functions  ┌--------┐
│                   │                   │steve   │
│                   │                   │michael │
│                   │                   │bob     │
│                   │                   └--------┘
│                   │
│                   ├---fst.fnc.gui-pres-common
│                   ├---fxt.fnc.os2pm-functions
│                   │      └--- ...
│                   └---fxt.fnc.osf-motif-functions
│                          └--- ...
└--adm.root ...
```

Figure 7. The TeamConnection Component Structure (Iteration 6)

Naming the Parts

Consider the following wish list items:

We expect to be able to:

- Have common and platform-specific source files that have the same name
- Allow controlled sharing of code among different products
- Build code for different target environments, using the same file names
- One user can work on multiple work areas or different projects or products at the same time on the same PC.
- WorkFrame projects can automatically be generated based on a TeamConnection work area.
- Build different "flavors" of the same executable for the same target platform:

All of the items listed above have one thing in common: the need to distinguish different files with the same name.

There is another important aspect of organizing parts when you think about using a common function or making an existing function a common function: **The naming convention for path names and file names should be standardized for all projects or products sharing code!**

A TeamConnection part name consists of the base name and the path name. We will introduce a familywide naming convention that enables us to distinguish:

- Common and platform-specific sources
- Platform-specific objects
- Different "flavors" of the same object code (for example, production code and code compiled for source level debug).

According to this naming convention, the first section of a part name specifies the platform:

com	for platform-independent code
os2	for OS/2-specific code
aix	for AIX-specific code
sun	for SUN-specific code

The platform section of the directory specification can then be followed by the "flavor." In our case, the following code "flavors" will be supported:

src	all the source code
obj	the production version of the object code
dbg	the same objects but compiled for source level debug.

Examples:

There is a common C-source *fstprog.c*, referencing the *fstiface.h* interface file as: `#include fstiface.h;`.
Because this interface file is common for OS/2 and AIX but has been modified for SUN, it exists as:

`comsrcfstiface.h` and `sunsrcfstiface.h`

The resulting AIX production version of the object will then have the name `aixobjfstprog.obj`, the corresponding version compiled for source level debug `aixdbgfstprog.obj`.

Note: It is recommended that you also use a naming convention for the parts identifying the project or product. This simplifies code sharing among different projects or products (remember the part names must be unique within a release) and simplifies the selection of parts belonging to a specific project or product or functional area.

In our case, we will use our project prefix, **fst**, for this purpose.

Examples:
`aixdbgfstprog.obj`, `comsrcfstiface.h`, `sunsrcfstiface.h`

Build Considerations

The TeamConnection build concept is based on three items:

- Build tree
- Builders
- Parsers

A TeamConnection build tree describes the dependencies of the parts. A build tree is constructed through the different connections (input, output, dependent) defined between parts.

A TeamConnection Builder is an object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers. A builder points to a build script that performs the steps in the build process. Use builders to define the specific tools that are needed for your development environment. Typically, your build administrator creates build scripts and builders, but anyone with the proper authority can do so.

For example, you are planning to build an executable program file from a set of C files. Your build process uses a particular compiler and linker. Therefore, you would create a builder that specifies that compiler and linker. Additionally, if you wanted to use a particular parser for your builds, you would create a parser object.

A superuser can check in files that are builder output. Once the builder output is checked in, it cannot be checked out. However, a superuser can replace the old builder output by checking in the new output.

A TeamConnection Parser is a tool that can read a source file and report back a list of dependencies of that source file. It frees a developer from having to know the dependencies one part has on other parts to ensure that a complete build is performed. For example, a C parser can read a C source code file and report back a list of the files included by the source file or by the included files.

You use the parse function to extract dependency information from build objects. The build tool uses this information to validate, maintain, and update interdependencies among build objects. When TeamConnection determines that a build object is out-of-date, the build tool invokes the parser you have defined for the object. The parser then does the following:

- Determines the dependencies that exist in the object being parsed
- Writes dependency information to a specified output file. An output file can only be an output for a single buildEvent.
- Opens and closes all input and output parts that might be associated with the parse operation

For example, suppose you have a C source part called MYFILE.C that contains the following statements:

```
#include <stdio.h>
#include <stdlib.h>
#include <mystuff.h>
```

The build invokes your parser (PARSER.EXE) on MYFILE.C and writes the output to a file called OUTPUT.TXT. OUTPUT.TXT would contain the following:

```
stdio.h
stdlib.h
mystuff.h
```

The build function would then do the following:

1. Read the output file
2. Compare the current set of dependencies with those returned by the parser
3. Add, delete, or update the dependencies for the object.

Note: The build function does not delete any dependencies that you add in the command line or GUI when you use the create or connect actions.

Note: Because the build script is copied to the build processor whenever necessary, keep this file small. Never specify the compiler (for example, ICC.EXE) as the build script, even if it is possible to do so.

- A compiler executable is usually a very large file, and the time needed to send it to the build processor will be an unnecessary overhead.
- If the compiler, for performance reasons, stays in memory even after the build is finished, a subsequent attempt of the build processor to copy the compiler again may fail, because the existing file is still “in use” by the system and therefore cannot be replaced.

Keeping Versions of Parsers and Builders

Even though TeamConnection itself does not provide special support to maintain versions of build scripts and parsers, it might be a good idea to keep versions of these programs with the project. Keeping versions will enable you to easily go back when there are problems and identify the changes that were made to these programs over time.

Because build scripts and parsers are really control elements and not the typical source or parts of your project, they should be kept separate from the other parts. Therefore the names of these build scripts and parsers should only contain the platform specification and no flavor specification.

Examples:
comfst_cparser.cmd, os2fst_cbuilder.cmd, aixfst_cbuilder.cmd

Because the builder is an implied dependency for any parts that use it, changing the build script always causes a rebuild of those parts. Unfortunately this will not be indicated in the build tree and will immediately affect all work areas of the release. There are two reasons why rebuild might not be what you want:

- You might want to run a test of the new builder first. Thus before you *activate* the new builder version, you want to see whether the builder works on all of the parts and the code compiled with the new builder runs as expected.
- Some changes to a build environment, such as moving the same version of a compiler to a different drive, should not trigger a rebuild, especially if you are dealing with large families.

Using *nested* build scripts with a primary command file being the TeamConnection build script (for example, :cbuilder.cmd; see Figure 8) and one or more secondary build scripts (for example, :os2\cbuilder.cmd; see Figure 9 on page 30) will do the job. If a change in a secondary build script triggers a rebuild, you have to check in the secondary build part to TeamConnection and connect it as an input to the parts using the builder. This setup will enable you to modify your build environment and test it in a work area before making it available to everyone.

```
cbuilder.cmd

  /** Simple Build script for C on OS/2 **/
   Parse Source myenv mycall mypname .
   Parse Value FileSpec("NAME",mypname) With myname "." .
   Parse Arg allargs
  "CALL os2\"myname  allargs
   Exit Rc
```

Figure 8. Primary Build Script

```
os2\cbuilder.cmd

  /** Simple Build script for C on OS/2 **/

   Rc = 99                     /* Be a pessimist !                      */

   my_options = "/c /DTRACE /DDEBUG /DFDEBUG",
               "/Fa+ /Fm- /Si+ /Sp1 /Ss /Su2 /Gd+ /Gm+ /J+ /Q+ /Ti+ /Ge+"
                               /*$-------------------------------------*\
                                  Get the call environment
                               \*--------------------------------------*/
   Parse Upper Source myenv mycall mypname .
   Parse Value FileSpec("NAME",mypname) With myname .
                               /*--------------------------------------*\
                                  Get arguments and environment info
                               \*--------------------------------------*/
   Parse Arg addargs

   tc_input  = Value("TC_INPUT",,"OS2ENVIRONMENT")
   tc_output = Value("TC_OUTPUT",,"OS2ENVIRONMENT")
                               /*--------------------------------------*\
                                  Remove own name from input list !
                               \*--------------------------------------*/
   inputs = tc_input
   tc_input = ""
   Do i=1 to Words(inputs)
     If (Pos(Word(translate(inputs),i),mypname) = 0)
       Then tc_input = tc_input Word(input,i)
     End i
```

Figure 9 (Part 1 of 2). Secondary Build Script

```
                             /*---------------------------------------*\
                                Setup process environment
                             \*---------------------------------------*/
”SET INCLUDE=tst\com;tst\os2;%include%”
                             /*---------------------------------------*\
                                It′s always good to show the arguments
                             \*---------------------------------------*/
Say myname”:”
 Say ”  Inputs...:” tc_input
 Say ”  Outputs..:” tc_output
 Say ”  Arguments:” add_args
                             /*---------------------------------------*\
                                Now run the process...
                             \*---------------------------------------*/
”ICC.exe /Fo”tc_output my_options add_args tc_input

 Exit Rc
```

Figure 9 (Part 2 of 2). Secondary Build Script

Using NULL Builders

Consider the following wish list item:

We expect to be able to:

- List all parts belonging to a functional unit

If all of the parts belonging to a functional component of your project are connected to a NULL builder, the NULL builder can then be used to selectively trigger the build of the specific functional component.

If you organize the build and packaging of your project using NULL builders that are attached to the functional component tree, you can easily find out which parts are contributing to a specific functional component. Simply run a report on the Build Part View (bPartView), using the name of the NULL builder in the -where ″nuPathName=′...′″ option:

```
TEAMC Report -raw -view bPartView
             -where ″nuPathName=′nullbuilderpart′″
```

Note: The same part may occur more than once in this report, because it can be a member of several subtrees of the build tree. You can easily sort out the duplicates, however, by using the build tree extraction tool, TWI_XBT.CMD (see “TWI_XBT: Extracting a Whole Build Tree” on page 151).

A NULL builder is commonly used as a collector object, what we would call a *pseudo target* in the old MAKE terminology. The benefit of using a NULL builder is that the input will not be transferred to the build processor during a build event. It will just be marked as successfully built by the build agent when it is encountered.

To use a NULL builder, you first have to create one. The key to creating one is in the -none and -script parameters. Make sure that you specify a type of -none and NULL for the -script parameter.

Here is an example:

```
  teamc builder -create MyNullBuilder
                -release prd.R001.00
                -script NULL
                -environment OS2
                -value 0
                -condition EQ
                -none
```

The -value and -condition parameters do not have any effect on NULL builders. Once you have a NULL builder, you can use it for any collector objects you create.

Chapter 4. Using TeamConnection and WorkFrame

There are two ways of using TeamConnection and WorkFrame together: You can use WorkFrame as a front-end tool to TeamConnection, integrating TeamConnection and WorkFrame actions, or you can use TeamConnection as a stand-alone library management tool, with no direct integration between TeamConnection and WorkFrame. In this chapter we describe both ways of using TeamConnection and WorkFrame together and explain the advantages and disadvantages of each.

The basic difference between the two methods is the build and how builds are done:

- When you use WorkFrame as a TeamConnection front end, the builds take place in TeamConnection using TeamConnection′s integrated build facility.
- When you use TeamConnection as a stand-alone library management tool, the builds are separated from each other. For the WorkFrame user, it will be business as usual, where edit, compile, and test are done using a "traditional" WorkFrame setup and only the "final" application build will be done using the TeamConnection build function.

WorkFrame As a TeamConnection Front End

When WorkFrame is used as a front-end tool to TeamConnection, all files are located in the TeamConnection database, and all builds are done using the TeamConnection build function. The basic characteristics of this method are:

- All data in the WorkFrame project is managed by TeamConnection.

- Builds are done remotely on the build servers.

- The functions invoked from WorkFrame are the functions provided by the TeamConnection command line interface or GUI.

Advantages

The advantages of using WorkFrame as a TeamConnection front end are:

Everything created in WorkFrame is visible to TeamConnection
TeamConnection can manage the versions of files and control file access.

Standardized build process
All compile and link options are controlled in one place by the TeamConnection build function.

Decrease in client processor use
Because compile and link are done on a remote build server, client processor use decreases. A fast build processor increases build performance.

Parallel build
You can build two or more parts simultaneously without using the resources on your own machine.

Independence from individual WorkFrame environments
You can work on different workstations with different WorkFrame setups or even at different locations, because the data and build settings are stored in the TeamConnection server.

Build independence of the target platform
By using the TeamConnection build function, you can build an application for OS/2, AIX, and MVS without leaving the WorkFrame environment.

Easy to find related parts for a change
A change to a common resource changes the status of related output parts. Therefore, by viewing the build tree, you can easily spot which parts have to be rebuilt.

Disadvantages

The disadvantages of using WorkFrame as a TeamConnection front end are:

Minimizes the flexibility of WorkFrame
You have to create different build trees depending on the compile options. For example, if for one build you need an option that creates a map file, you need a different build tree (even though this disadvantage can be limited by the use of environment variables and setting the variables at build time by using the *-parameters* attribute).
You have to change the build tree if the dependencies change. For example, if you decide to split one source into two different sources, you have to update the build tree manually. In a stand-alone WorkFrame environment, the makemake utility makes these changes automatically. But if you only add a new include statement in the source, the parser will find the dependency and automatically update the build tree. The included part has to be created in TeamConnection.

Degrades performance
A single compile might be slower because of the communication overhead between TeamConnection and the build processor. Editor load time might be slower if a part has to be checked out first.

TeamConnection As a Stand-Alone Library Management Tool

When you use TeamConnection as a stand-alone library management tool, editing, compiling, and testing are done locally in the WorkFrame environment:

- The basic settings do not change from the VisualAge C++ project. Action, variable, and type settings are inherited from the VisualAge C++ project.
- You can check in the files to TeamConnection whenever you need to.
- Production build takes place in a TeamConnection environment.

Advantages

The advantages of using TeamConnection as a stand-alone library management tool are:

- **Retain the flexibility of WorkFrame**

  You can change compile and link options easily and add or delete files without having to think about the build structure.
- **Same user interface for existing WorkFrame users**

  For existing WorkFrame users, use of the product does not change.

Disadvantages

The disadvantages of using TeamConnection as a stand-alone library management tool are:

- **You are responsible for synchronizing parts**

  You have to check in all WorkFrame files to TeamConnection when you have completed your work.
- **You are responsible for synchronizing build environment**

  You have to set up the same build environment in TeamConnection as in the local WorkFrame project. That is, compile and link options must be changed to the local WorkFrame settings before the production build can be executed.
- **You have to work on the same workstation**

You cannot change between different workstations, because the WorkFrame environment may differ between the various machines.

Our Approach

As a starting point, we decided to use WorkFrame as a TeamConnection front-end tool and then implement VisualAge for C++ functions into WorkFrame to create a realistic and workable application development environment.

Chapter 5. Setting Up WorkFrame

TeamConnection provides a customized WorkFrame project for integration with WorkFrame that enables you to use WorkFrame as a front-end tool. However, this project does not provide the integrated environment we expect primarily because it starts up the TeamConnection GUI client for every TeamConnection action that is called from within the WorkFrame project. For some actions, this is redundant from a user interface point of view. In addition you have to change the TeamConnection GUI client settings each time you work on a different project or product. So we decided to create our own project setting, called TWI_PRJ, to better integrate TeamConnection and WorkFrame. In this TWI_PRJ project, we defined actions that invoke a REXX program, using the TeamConnection command interface to reduce the redundancy. We also created an installation script for Project Smarts to set up the TWI_PRJ project automatically from the TeamConnection work area.

In this chapter, we describe how to create a WorkFrame project from the TeamConnection work area in general and set up and customize the TWI_PRJ project.

See *VisualAge C++ for OS/2 User's Guide Version 3.0*, S25H-6961, for more details regarding the WorkFrame project setup.

Creating WorkFrame Project from TeamConnection Work Area

Consider the following wish list item:

We expect that:

- One user can work on multiple work areas or different projects or products at the same time on the same PC.

Before creating a WorkFrame project, you must create a work area in TeamConnection. Then you can create a WorkFrame project that represents the TeamConnection work area.

After you have created the work area, you have to check out the parts that have to be included in the WorkFrame project and updated in your work area. If you are using a TeamConnection client, it does not matter which directory TeamConnection uses as long as it is a working directory. If you are using WorkFrame, however, you must know to which directory the parts were checked out; otherwise, you will not be able to see the parts that you want to work on from within the WorkFrame project.

TeamConnection checks out the parts to the directory you specified in the relative name field if you used the command line interface or in the destination directory field if you used the TeamConnection GUI. If you work with more than one work area, you must specify a different directory because you might have the same part name in different work areas. Therefore your main focus should be to distinguish parts with the same name from each other. To make a part unique within your workstation, you have to embed the family name, release name, and work area name in the part name because the parts are unique within a work area, the work area is unique within a release, and the release is unique within a family.

We decided to make a directory and check out the required parts to this directory:

```
base-directoryfamily-nameworkarea-namerelease-name
```

where:

base-directory Can be any directory path in accordance with your own preference.

We realized that we had to check out not only the parts that we were editing but also the parts that we were building because, to start a TeamConnection build process, we had to specify the name of the *target* part that had to be built. In addition, to get the *target* part name into WorkFrame, the part had to exist on the same workstation as the workstation on which we were working.

Note: If you use the VisualAge C++ build function in WorkFrame, which is mainly the local build function, you also have to check out all dependent parts. You do not have to check out dependent parts if you use our TWI_PRJ installation script, which extracts all parts required to build a target part.

TWI_PRJ Installation Script

The TWI_PRJ installation script, TWI_PRJ.CMD, automatically creates a TWI_PRJ. See "TWI_PRJ: Creating the WorkFrame Project" on page 128 for the REXX source.

TWI_PRJ.CMD sets up the project as follows.

1. **Creates a project icon**

   The script creates a TWI_PRJ project icon called *release-name workarea-name* in the folder you specify. You can change the name of the folder.

2. **Creates the working directory for the project**

   The script creates a working directory, `family-nameworkarea-namerelease-name`, under the directory that you specify. Periods (.) and back slashes () used in the name are converted to hyphens (-).

3. **Sets up the project environment variables in the project′s tools setup**

   TC_FAMILY
   Specifies the family name to be used in the project. This is mandatory.

   TC_RELEASE
   Specifies the release name to be used in the project. This is mandatory.

   TC_WORKAREA
   Specifies the work area name to be used in the project. This is mandatory.

   TC_USER
   Specifies the user ID to be used in the project. This is mandatory.

   TC_BECOME
   The user ID used to access objects in TeamConnection. It may differ from TC_USER. This environment variable is optional.

   TC_CASESENSE
   Specifies whether the case of the arguments in commands are changed to the upper case or lower case or not changed. You specify *UPPER*, *LOWER*, or *IGNORE*.

   TC_BUILDPOOL
   Specifies the build pool name to be used in the project. This is mandatory.

   TWI_RELATIVE
   Used as relative option name for each TeamConnection command in our REXX program. The value is the same as the working directory. The default value is *drive:family-nameworkarea-namerelease-name*
   .

TWI_TARGET_PART
: Specifies the target part name used as the target file name in the project settings. You provide the name. This parameter is used on the TC Build Target and TC Build Target Forced actions.

TWI_TRACE
: If the value is *ON*, the trace is turned on for each action.

4. **Sets up the actions in the project′s tools setup**

    - TC Edit
    - TC CheckIn Forced
    - TC CheckIn
    - TC CheckOut
    - TC Lock
    - TC UnLock
    - TC Extract
    - TC Touch Parts
    - TC Build Part
    - TC Build Part Forced
    - TC Build Target
    - TC Build Target Forced
    - TC Show WorkArea
    - TC Freeze WorkArea
    - TC Task List

    See "Actions Defined in the TWI_PRJ Project" on page 75 for details of the actions.

5. **Checks out the parts and extracts other required parts**

    The parts that you select are checked out to the working directory. All other parts that belong to the build tree of the target part are extracted. Extracted source parts are changed to read-only files.

Adding Our Project Install Script to Project Smarts

Consider the following wish list item:

We expect that:

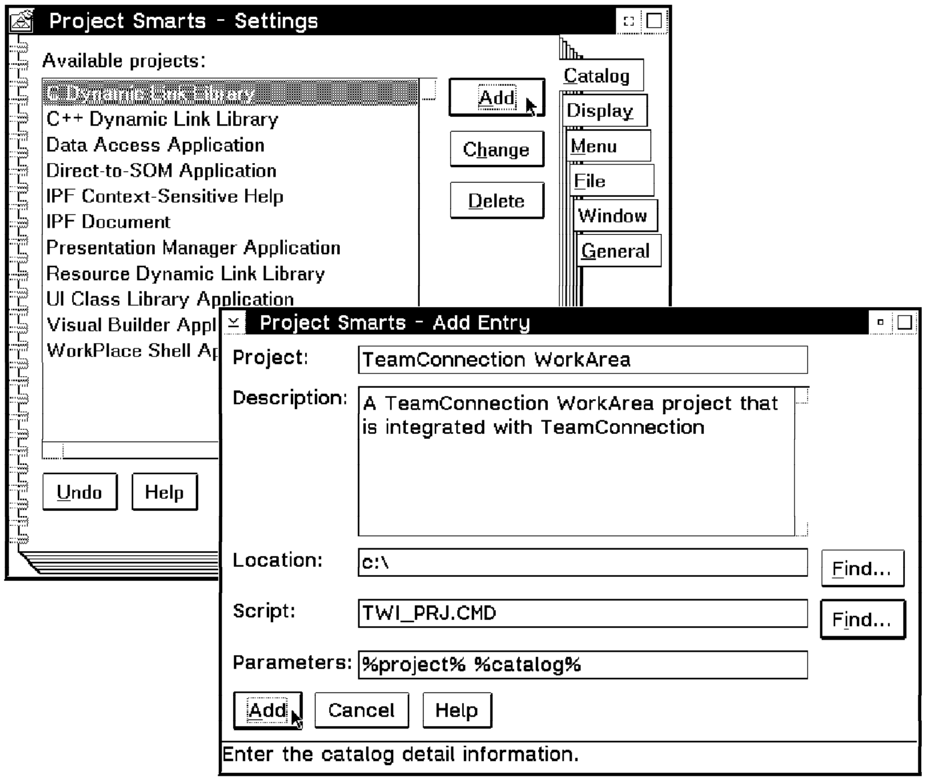
- WorkFrame projects can automatically be generated based on a TeamConnection work area.

To create a TWI_PRJ project using Project Smarts, you must first add the project installation script, TWI_PRJ.CMD, to the Project Smarts catalog (see Figure 10 on page 44).

- Open the Project Smarts – Settings window.
- Select **Add**.
- On the Project Smarts – Add Entry window enter **TeamConnection Work Area** for the project name.
- You can enter any text in the Location field because it will not be used.
- Enter **TWI_PRJ.CMD** for the script name.
- Enter **%project% %catalog%** for the parameters.

  You can also add the **trace** option as a parameter to enable the generation of additional output for debug purposes.
- Select **Add**.

.*



Project: TeamConnection WorkArea

Description: A TeamConnection WorkArea project that is integrated with TeamConnection

Location: c:\

Script: TWI_PRJ.CMD

Parameters: %project% %catalog%

Enter the catalog detail information.

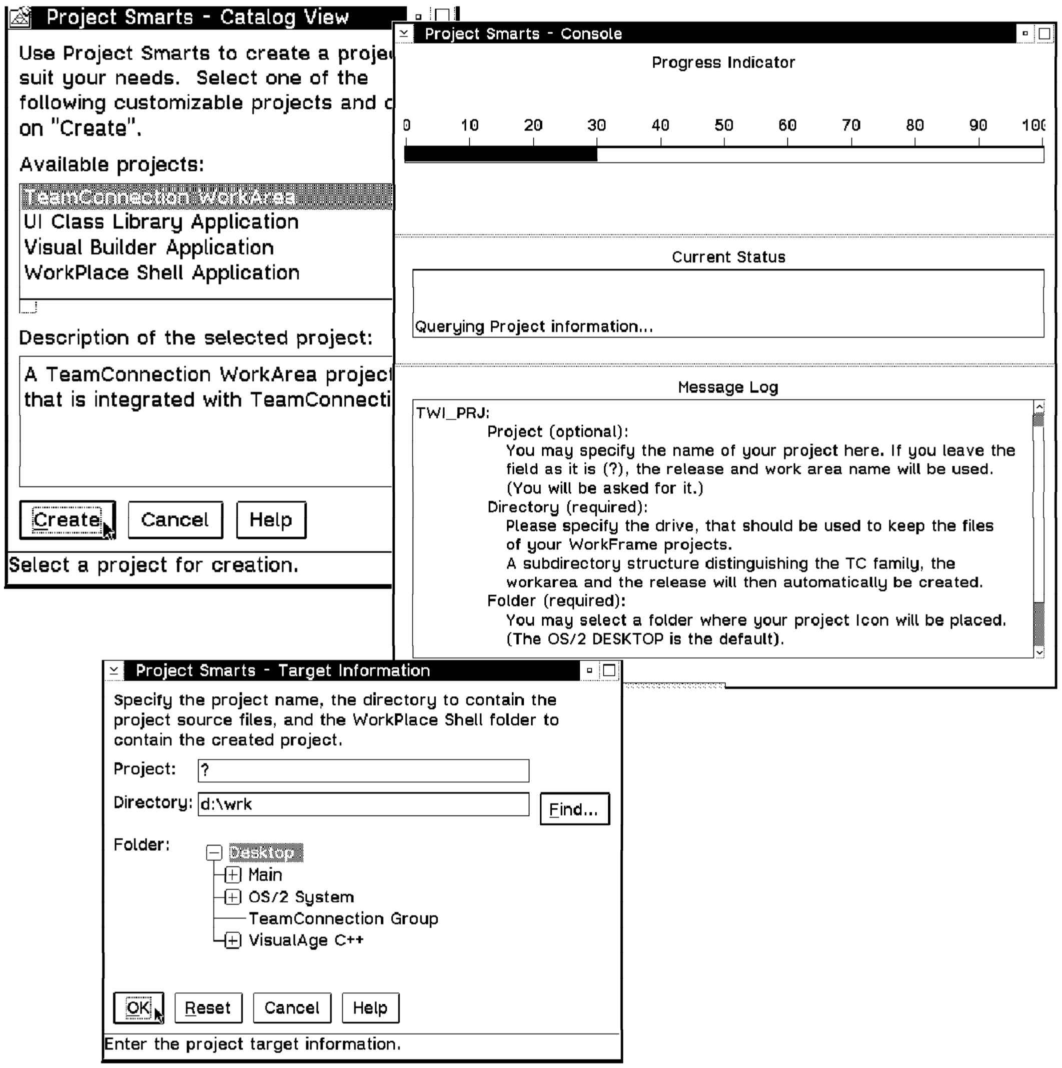
Figure 10. Adding the Project Installation Script

Creating WorkFrame Project Using Project Smarts

Once you have added the script to your Project Smarts catalog, you can create a WorkFrame project from Project Smarts (see Figure 11 on page 46):

- Double-click on the **Project Smarts** icon to start Project Smarts.
- Select **TeamConnection Work Area** from the available projects.
- Select **Create**.
- On the Project Smarts – Target Information window change the target information fields according to the message log on the Project Smarts – Console window.

Project	Fill in the project name. The default is `workarea(release)`
Directory	Type the base working path name. You must specify at least the drive. The working directory is `basefamilyworkarearelease`
Folder	Select the folder that will contain the created project icon.


Project Smarts - Catalog View

Use Project Smarts to create a proje
suit your needs. Select one of the
following customizable projects and c
on "Create".

Available projects:

TeamConnection WorkArea
UI Class Library Application
Visual Builder Application
WorkPlace Shell Application

Description of the selected project:

A TeamConnection WorkArea project
that is integrated with TeamConnecti

Select a project for creation.

Project Smarts - Console

Current Status

Querying Project information...

Message Log

```
TWI_PRJ:
        Project (optional):
           You may specify the name of your project here. If you leave the
           field as it is (?), the release and work area name will be used.
           (You will be asked for it.)
        Directory (required):
           Please specify the drive, that should be used to keep the files
           of your WorkFrame projects.
           A subdirectory structure distinguishing the TC family, the
           workarea and the release will then automatically be created.
        Folder (required):
           You may select a folder where your project Icon will be placed.
           (The OS/2 DESKTOP is the default).
```

Project Smarts - Target Information

Specify the project name, the directory to contain the project source files, and the WorkPlace Shell folder to contain the created project.

Project: ?

Directory: d:\wrk

Folder:
- Desktop
  - Main
  - OS/2 System
  - TeamConnection Group
  - VisualAge C++

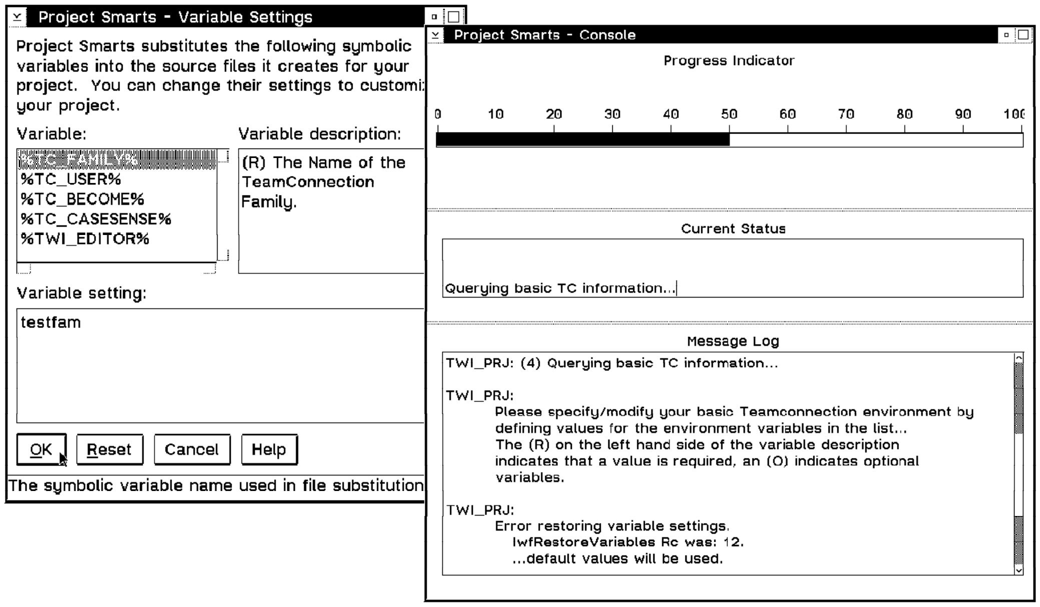
Enter the project target information.

Figure 11. Creating a TWI_PRJ Project from Project Smarts (1)

Setting the Basic TeamConnection Variables

In the Project Smarts - Variable Settings window, change the basic TeamConnection variables to the values you want (see Figure 12).

Note: See "TWI_PRJ Installation Script" on page 41 or the Variable description field for the value of the variables. Variables with (R) are mandatory; variables with (O) are optional.



Project Smarts substitutes the following symbolic variables into the source files it creates for your project. You can change their settings to customi: your project.

Variable:
%TC_FAMILY%
%TC_USER%
%TC_BECOME%
%TC_CASESENSE%
%TWI_EDITOR%

Variable description:
(R) The Name of the TeamConnection Family.

Variable setting:
testfam

The symbolic variable name used in file substitution

Current Status: Querying basic TC information...

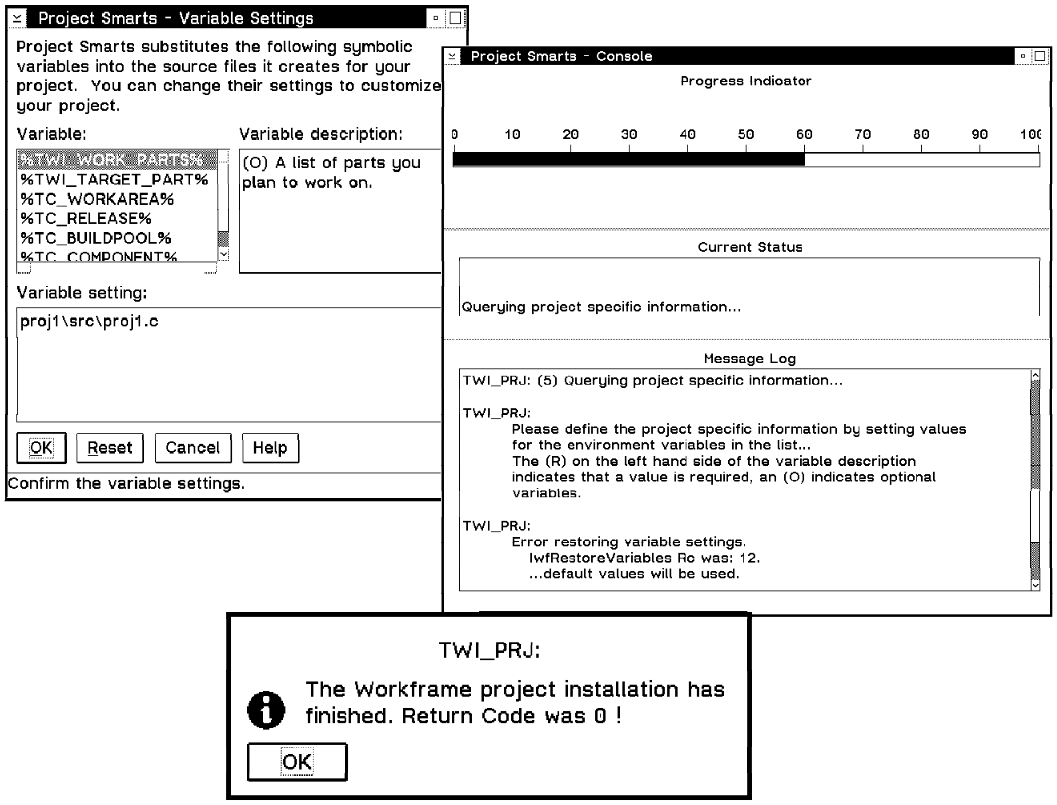
```
TWI_PRJ: (4) Querying basic TC information...

TWI_PRJ:
        Please specify/modify your basic Teamconnection environment by
        defining values for the environment variables in the list...
        The (R) on the left hand side of the variable description
        indicates that a value is required, an (O) indicates optional
        variables.

TWI_PRJ:
        Error restoring variable settings.
          IwfRestoreVariables Rc was: 12.
          ...default values will be used.
```

Figure 12. Creating a TWI_PRJ Project from Project Smarts (2)

Setting the Work Area Specific Variables

You can change the work area specific variables to the values you want in the Project Smarts - Variable Settings window (Figure 13).

Select **OK** when the message box appears.



Project Smarts substitutes the following symbolic variables into the source files it creates for your project. You can change their settings to customize your project.

Variable:	Variable description:
%TWI_WORK_PARTS%	(O) A list of parts you plan to work on.
%TWI_TARGET_PART%	
%TC_WORKAREA%	
%TC_RELEASE%	
%TC_BUILDPOOL%	
%TC_COMPONENT%	

Variable setting:

```
proj1\src\proj1.c
```

Confirm the variable settings.

Querying project specific information...

```
TWI_PRJ: (5) Querying project specific information...

TWI_PRJ:
        Please define the project specific information by setting values
        for the environment variables in the list...
        The (R) on the left hand side of the variable description
        indicates that a value is required, an (O) indicates optional
        variables.

TWI_PRJ:
        Error restoring variable settings.
          IwfRestoreVariables Rc was: 12.
          ...default values will be used.
```

TWI_PRJ:

The Workframe project installation has finished. Return Code was 0 !

Figure 13. Creating a TWI_PRJ Project from Project Smarts (3)

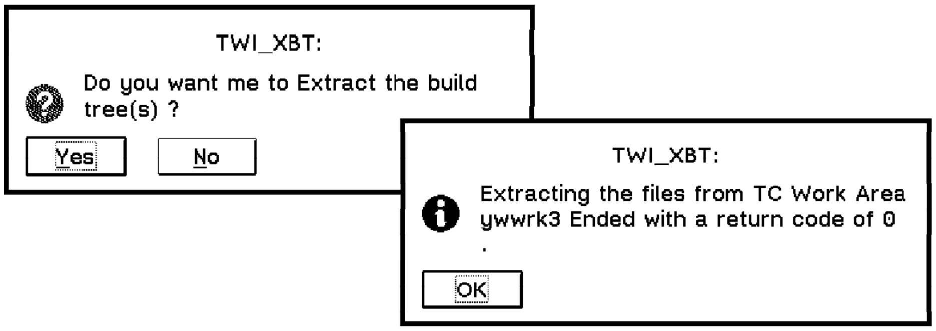
Automatic Extraction of the Build Tree

If there is no file or subdirectory in the `base\family\workarea\release` working directory, TWI_PRJ.CMD will call the automatic extraction program, TWI_XBT.CMD, to extract parts that you need in the project. The extraction is asynchronous, so even though the project has been created, the extraction may still be in progress. TWI_XBT.CMD extracts the parts in the following manner (see "TWI_XBT: Extracting a Whole Build Tree" on page 151 for the source code):

1. If the %TWI_WORK_PARTS% is specified, TWI_XBT.CMD uses this as a *parts list*.
2. If the %TWI_WORK_PARTS% is not specified, TWI_XBT.CMD invokes the TeamConnection client and lets you extract the parts you need to the working directory. Then TWI_XBT.CMD uses the files in the working directory as the *parts list*.
3. For each part in the parts list, TWI_XBT.CMD looks for the closest parent part with an associated builder.
4. For each parent part that it finds, TWI_XBT.CMD extracts all of the parts in its build tree.

If you want to use automatic build tree extraction:

Select **Yes** when the *Do you want me to Extract the build tree(s)* message appears (see Figure 14).
Select **OK** when the message box appears.


TWI_XBT:

Do you want me to Extract the build tree(s) ?

TWI_XBT:

Extracting the files from TC Work Area ywwrk3 Ended with a return code of 0 .

Figure 14. Creating a TWI_PRJ Project from Project Smarts (4)

Customizing the Settings of the TWI_PRJ Project

In this section we show you step by step how to customize the settings in the TWI_PRJ project.

Setting the Project Location

By now you will be able to see the project icon on your desktop or in whichever folder you specified. Open the project view and check whether all parts are visible from the WorkFrame. If you do not have any paths in your part name, you will see the parts. If you have paths, as our project or product has, you will not see any parts in your project. To see the parts, you must change the location settings of your project to make files visible from the project. On the TWI_PRJ - Icon view window:

- Open the project settings by selecting **View** → **Settings** → **Location**.
- Select **Find**.
- Select the desired directory and select **Add**.
- Add all directories and select **OK**.
- Close the Settings window.

Now you should see all of the parts that have been extracted.

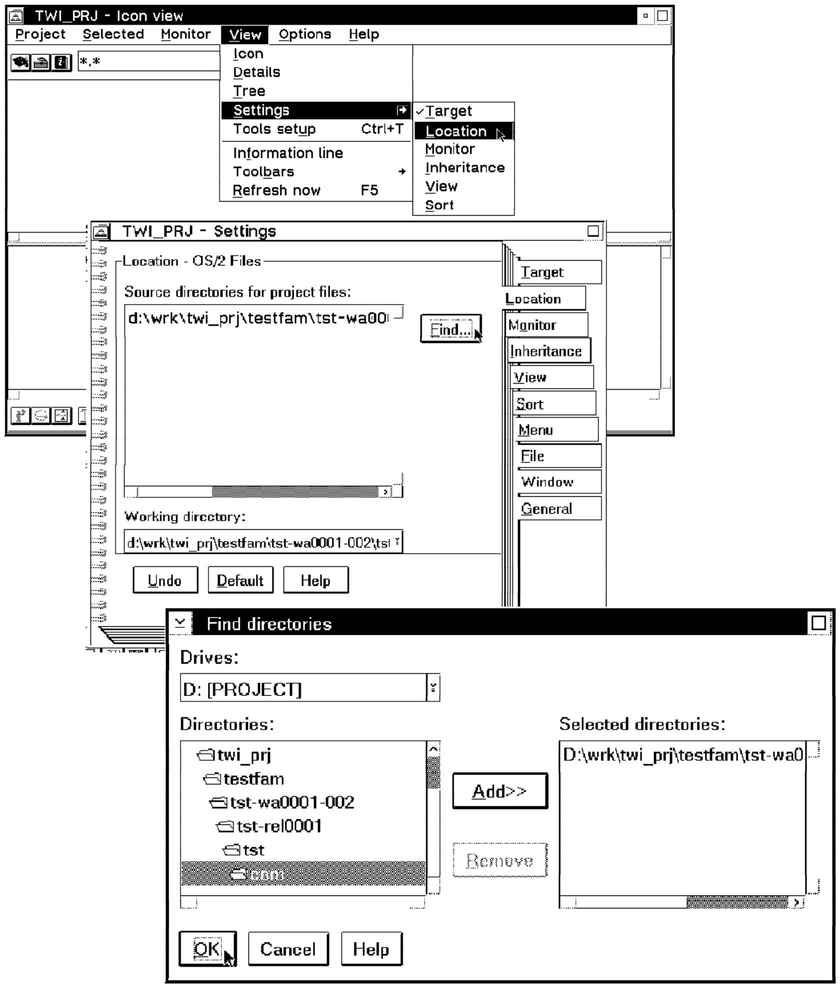


Figure 15. Changing the Settings of TWI_PRJ

Changing the Default Editor from LPEX to EPM

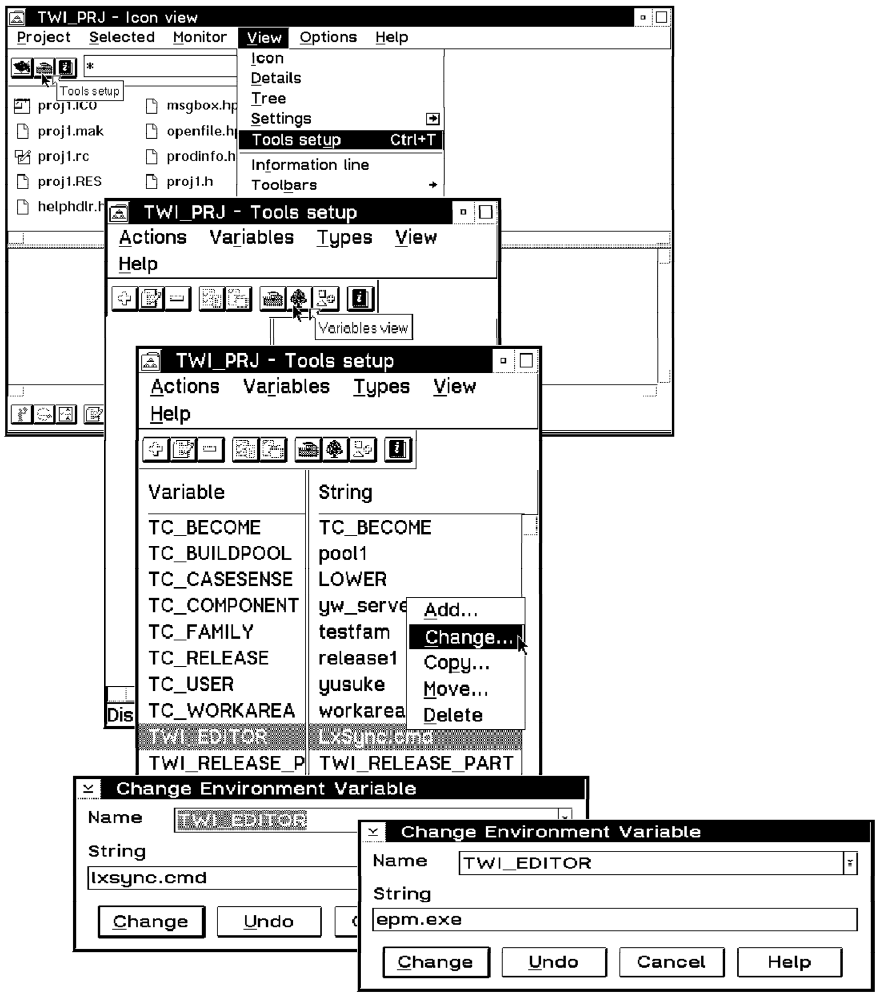
In this section we show you how to change the default editor settings from LPEX to, for example, EPM. We also show you how to change the action options.

Setting the TWI_EDITOR variable

You may want to change the default editor of the TC Edit action from, for example, LPEX (which we call using the command file, LXSYNC; see "LXSYNC: A Synchronous Way of Invoking LPEX" on page 190) to another editor such as EPM:

- From the TWI_PRJ - Icon view window (Figure 16 on page 53) select the Tools setup icon or select **Tools setup** from the View pull-down menu.
- On the TWI_PRJ - Tools setup window select the Variables view icon.
- Click with mouse button 2 on the TWI_EDITOR variable to display the pop-up menu.
- Select **Change**.
- In the Change Environment Variable window change the String field to the name of the command file or program you use to invoke your editor; for example, **epm.exe**.
- Select the **Change** push button.

Note: LXSYNC.CMD is the command file that will invoke LPEX synchronously (see "LXSYNC: A Synchronous Way of Invoking LPEX" on page 190).



Variable	String
TC_BECOME	TC_BECOME
TC_BUILDPOOL	pool1
TC_CASESENSE	LOWER
TC_COMPONENT	yw_serve
TC_FAMILY	testfam
TC_RELEASE	release1
TC_USER	yusuke
TC_WORKAREA	workarea
TWI_EDITOR	LxSync.cmd
TWI_RELEASE_P	TWI_RELEASE_PART

Name: TWI_EDITOR
String: lxsync.cmd

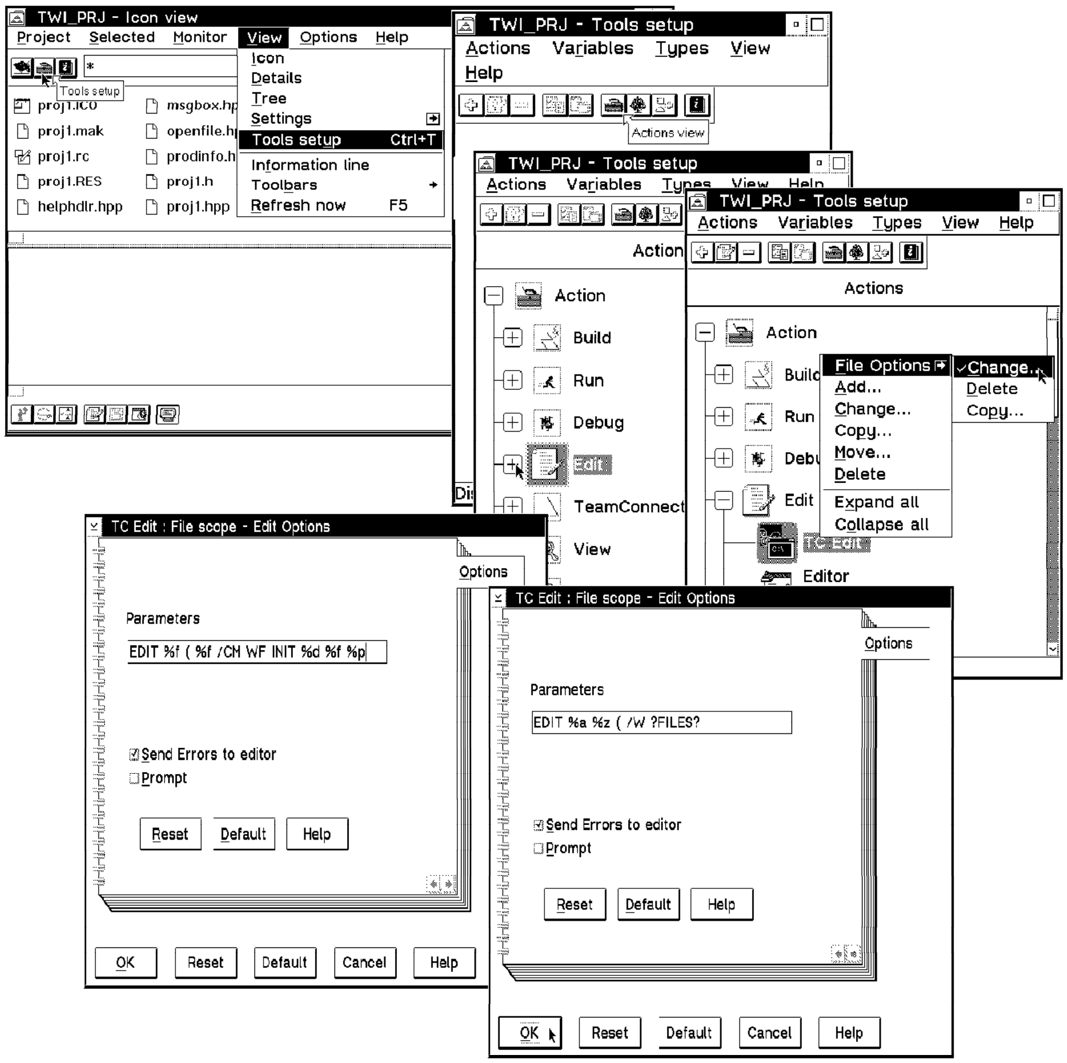
Name: TWI_EDITOR
String: epm.exe

Figure 16. Changing the Default Editor

Changing the TC Edit Action Options for EPM

As you have changed the editor to be invoked, you must also change the options of the action (see Figure 17 on page 55):

- On the TWI_PRJ - Tools setup window, select the Actions view icon.
- Expand the Edit class.
- Click with mouse button 2 on **TC Edit action** to display the pop-up menu.
- Select **File Options** → **Change**
- Change the options of the action:
  ```
  EDIT %a %z ( /W ?FILES?
  ```



Parameters: EDIT %f (%f /CH WF INIT %d %f %p

[✓] Send Errors to editor
☐ Prompt

Parameters: EDIT %a %z (/W ?FILES?

[✓] Send Errors to editor
☐ Prompt

Figure 17. Changing the TC Edit Action Options

The Options of TC Edit Action for EPM and LXSYNC

The options of TC Edit action are separated by a left parenthesis ((). Options before the parenthesis describe the action and the options used for the checkin and checkout functions. Options after the parenthesis are used for the editor.

- For EPM.EXE, the option for the editor itself is `/W %a %z`. Checkin and checkout require the file names to be passed and the options will be `%a %z`. So the option for the TC Edit action will be : `EDIT %a %z ( /W %a %z`.

  If we specify this in the Parameters field on the TC Edit: File Scope – Edit Options page, the system hangs. So to avoid this, we use a variable, `?FILES?`, which is translated to `%a %z` when the editor is called.

- For LXSYNC.CMD, the option for the editor itself is `%f /CM WF INIT %d %f %p`. Because the editor is invoked for one file at a time, the option for checkin and checkout is `%f`. So the Edit Options parameters for TC Edit will be `EDIT %f ( %f /CM WF INIT  %d %f %z`.

Changing the Applicable Actions on Files

As you can see in Figure 18, the applicable actions differ according to the type of file selected because the actions have different applicable types in the action settings. For example, if you see the types menu in the action definition of *TC Edit*, the source types are set to **Editable**. For the *TC Build Part* action the source types are set to **NotTCSaved**.

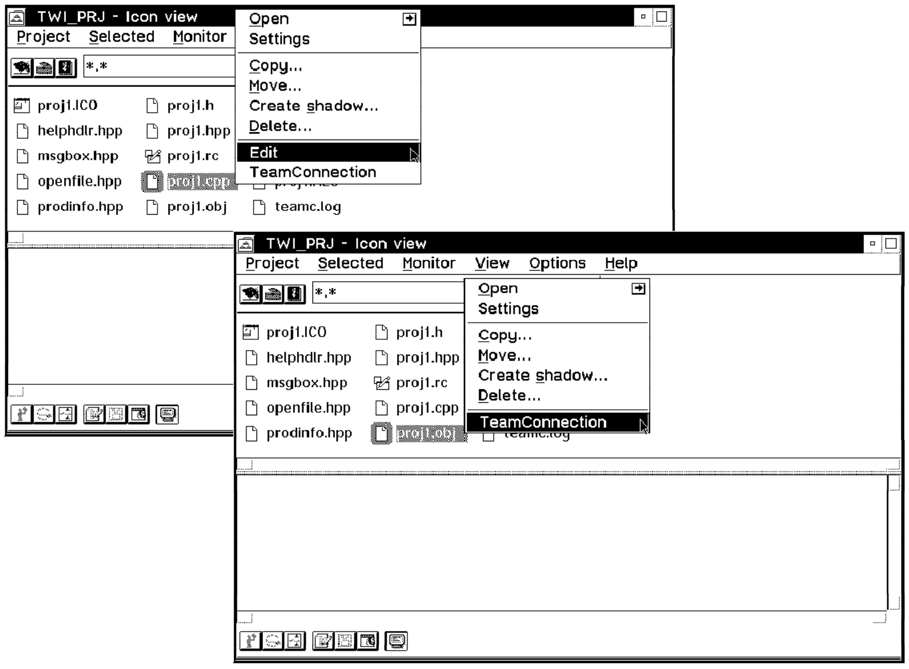


Figure 18. Applicable Actions on Files

To change the applicable actions for various file types do the following:

- Open the TWI_PRJ - Tools setup window (Figure 19).
- Select the Actions view icon.
- Click with mouse button 2 on **TC Edit Part** action.
- Select **Change**.

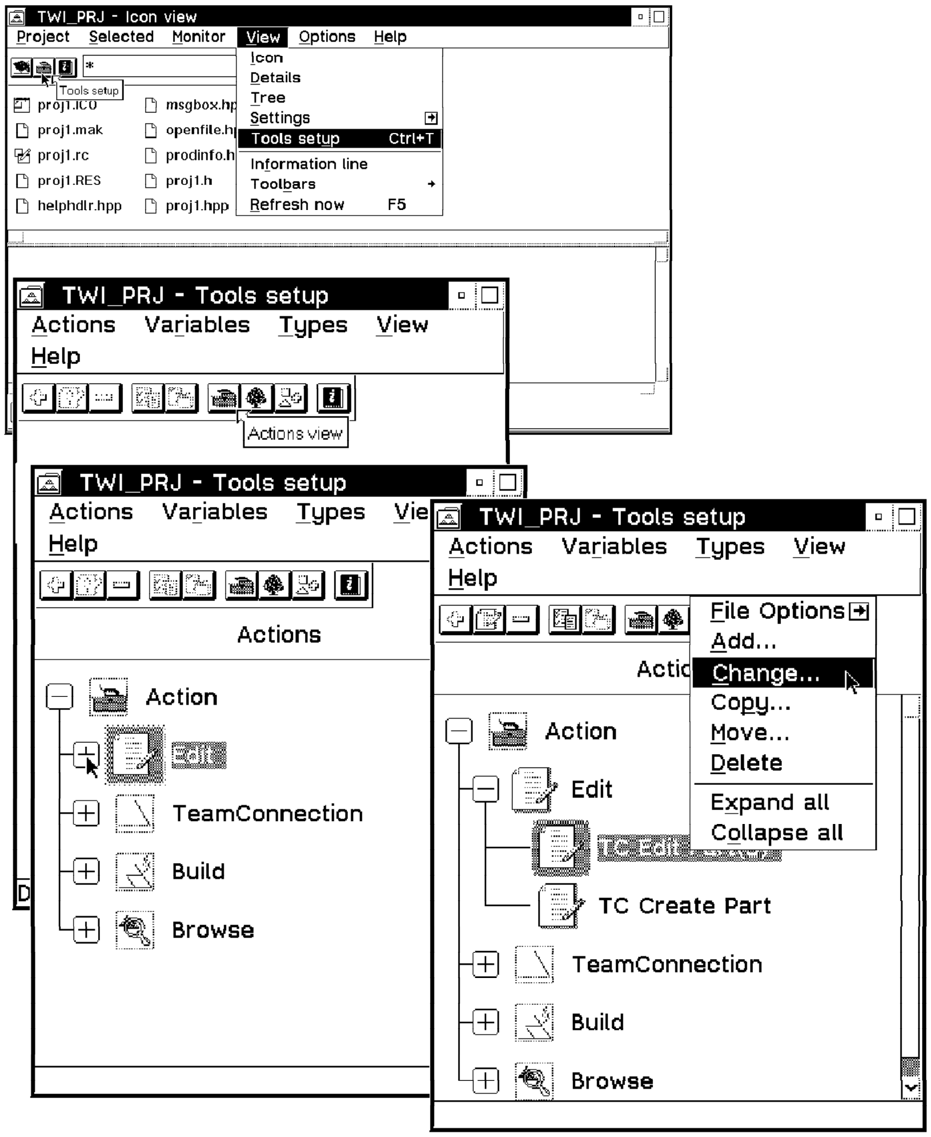
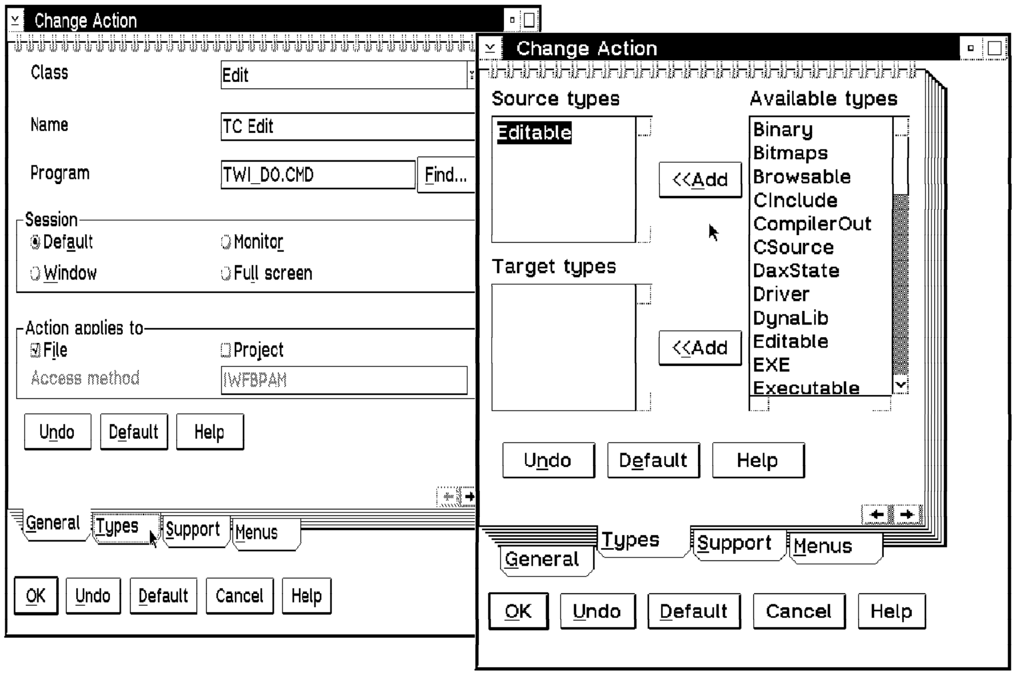


Figure 19. Types of Files Set in the Action Definition (1)

- On the Change Action window, select the **Types** menu (see Figure 20 on page 60).
- Select the **Types** for the TC Build part action as well.

...


Change Action

Class: Edit

Name: TC Edit

Program: TWI_DO.CMD

Session: [✓ Default] Monitor Window Full screen

Action applies to: [✓ File] Project

Access method: IWFBPAM

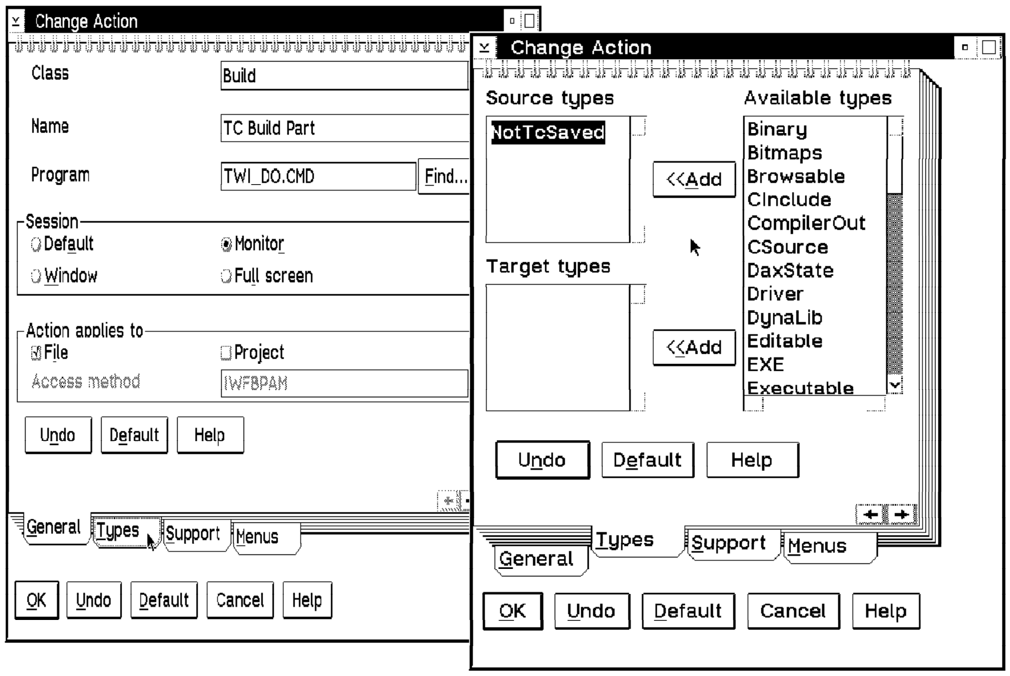
Change Action

Source types: Editable

Target types:

Available types: Binary, Bitmaps, Browsable, CInclude, CompilerOut, CSource, DaxState, Driver, DynaLib, Editable, EXE, Executable

...


Change Action

Class: Build

Name: TC Build Part

Program: TWI_DO.CMD

Session: Default [✓ Monitor] Window Full screen

Action applies to: [✓ File] Project

Access method: IWFBPAM

Change Action

Source types: NotTcSaved

Target types:

Available types: Binary, Bitmaps, Browsable, CInclude, CompilerOut, CSource, DaxState, Driver, DynaLib, Editable, EXE, Executable

Figure 20. Types of Files Set in the Action Definition (2)

If you want to make an action applicable to another type of file, change the source type by adding or overwriting it. Or add a new type in the Types view and add it to the action settings (see “Adding Our Project Install Script to Project Smarts” on page 43).

Changing the Default Actions

The order of the actions in the menu represents the priority of the actions. The top action will be the default action when you double-click on a file. In Figure 18 on page 57, the default action for the C++ source file is **TC Edit** and for the object file it is **TC Build Part**. So when you double click on a C++ source file, the editor comes up containing the file.

Say you want to change the default action of the source file from TC Edit to TC CheckOut. To do so, make the priority of TC CheckOut action higher than TC Edit:

- Open the TWI_PRJ - Tools setup window (Figure 21 on page 63)
- Select the Actions view icon.
- Click with mouse button 2 on the TC CheckOut action.
- Select **Change**.
- Select the Support menu (Figure 22).
- Change the **Priority** to a number higher than the TC Edit Part action.

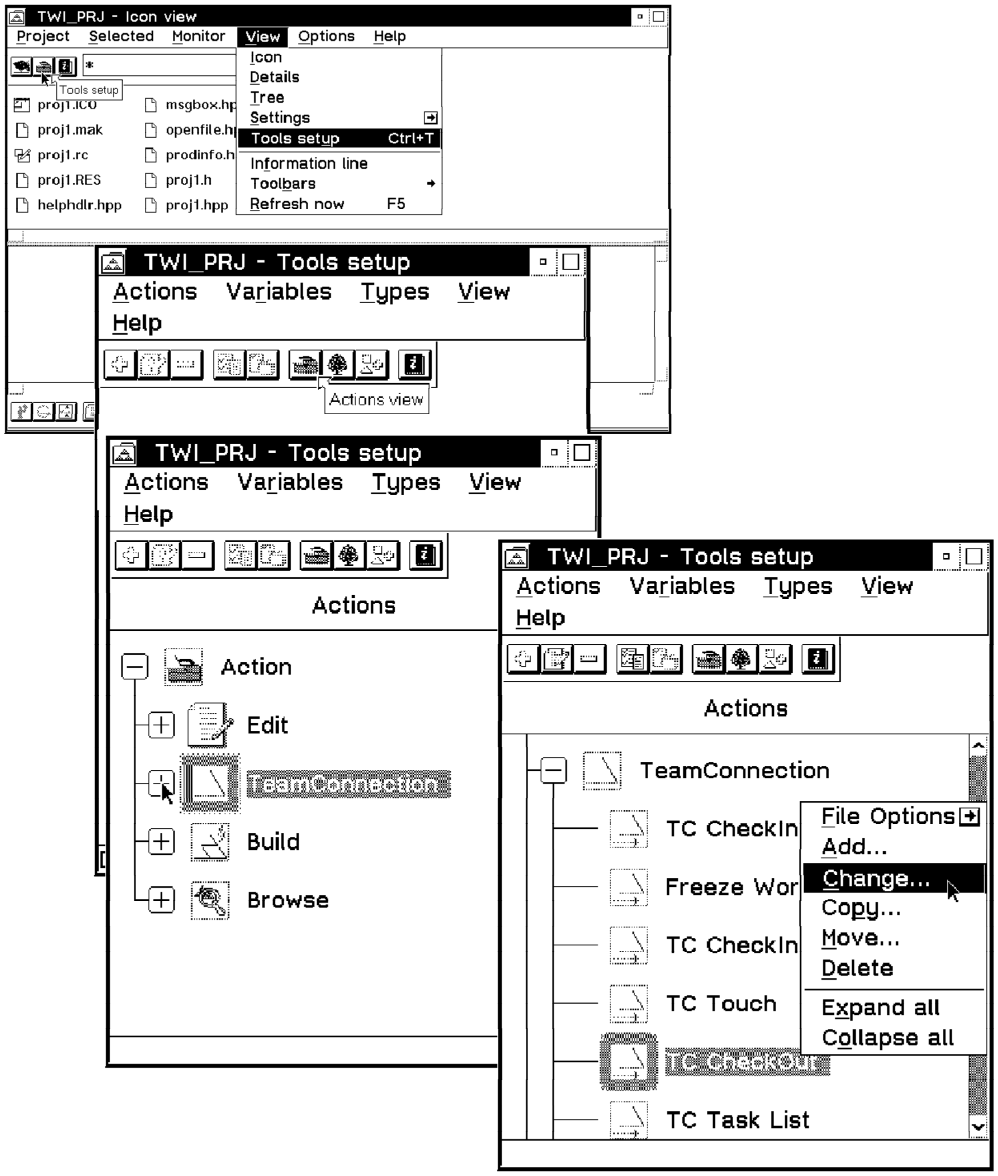
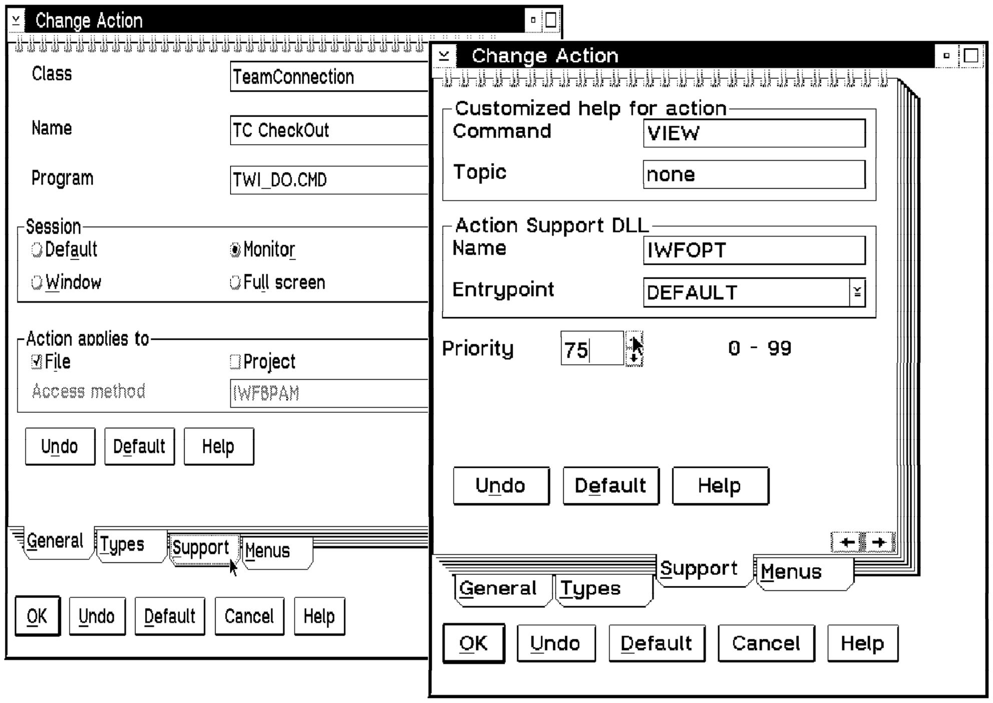


Figure 21. Changing the Priority of an Action (1)


Change Action

Class: TeamConnection

Name: TC CheckOut

Program: TWI_DO.CMD

Session: Default, Window, [✓ Monitor], Full screen

Action applies to: [✓ File], Project

Access method: IWFBPAM

Change Action

Customized help for action

Command: VIEW

Topic: none

Action Support DLL

Name: IWFOPT

Entrypoint: DEFAULT

Priority: 75 0 - 99

Figure 22. Changing the Priority of an Action (2)

Adding a New Environment Variable

You may find that you need other environment variables that have not been previously defined. For example, you may want to set *d:\common\inc* as an include directory.

To add a new environment variable:

- Switch to the Variables view and click on the Add icon (Figure 23).
- In the Add Environment Variable window type **include** in the Name field.
- Type **d:commoninc** for the string.
- Select **Add**.

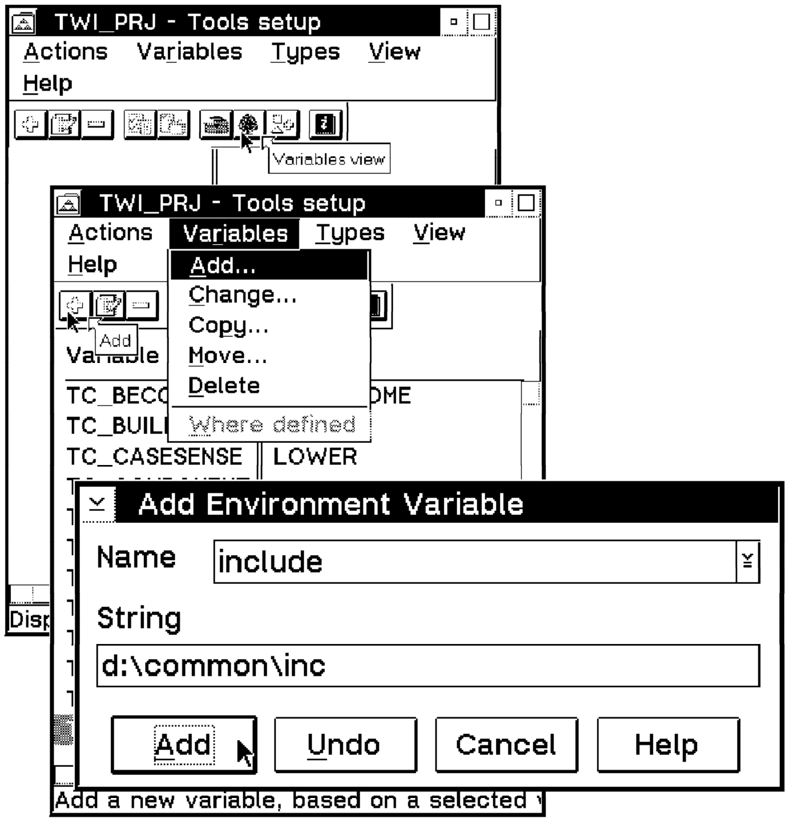


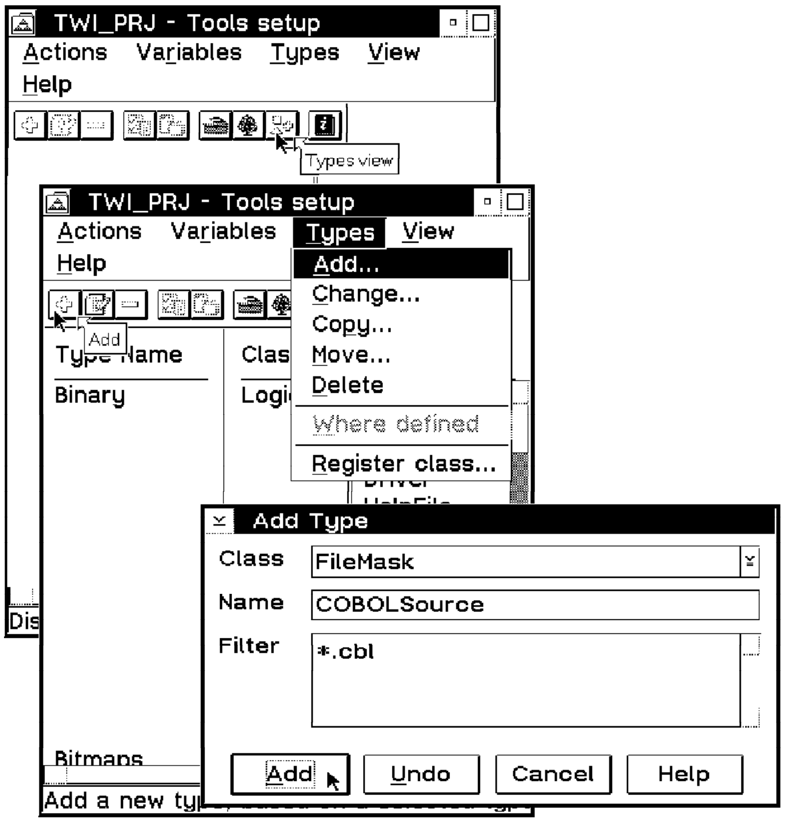
Figure 23. Adding a New Environment Variable

Adding a New Type

Types are used to distinguish a group of files from other files and to apply actions to specific files. For example, you might create a file type called *COBOL Source* to apply a COBOL compiler action to files that have a *.cbl* extension.

To add a new type:

- Switch to the Types view and click on the Add icon (Figure 24).
- In the Add Type window, select **FileMask** (from the Class field's selections) for the Class field.
- Add a new Type **COBOLSource** in the Name field.
- Type ***.cbl** in the Filter field.
- Select **Add**.



Class: FileMask
Name: COBOLSource
Filter: *.cbl

Figure 24. Adding a New Type

General Customization of WorkFrame Projects

In this section we take a closer look at the general customization of a project in WorkFrame. We show you how to change the project settings and add an action.

Changing the Project Settings

In the project settings, you would mainly change the target, location, monitor, and inheritance.

When using TWI_PRJ **do not** change the working directory in the location menu. If you change it, some of the actions will not be available.

Inheritance is important because the actions, variables, and types also change. If there is a setting in your project that has the same name as the project from which you are inheriting, the setting will not be inherited. For example, if you already have a TC_FAMILY variable defined, you cannot inherit another TC_FAMILY variable. Also, if you have an action called *Edit*, you cannot inherit another action called *Edit* even if it has different settings. If you want to change the settings that you inherited, you must change the settings in the original project.

Figure 25 on page 68 shows an example of customizing project settings. For each Settings page, we explain the various fields on the page:

Settings – Target page

- Specify the target file name in the Name field on the Target page.
- Specify the run option in the Run options field on the Target page.
- Specify the make file name (not needed for our project) in the Makefile field on the Target page.

Settings – Location page

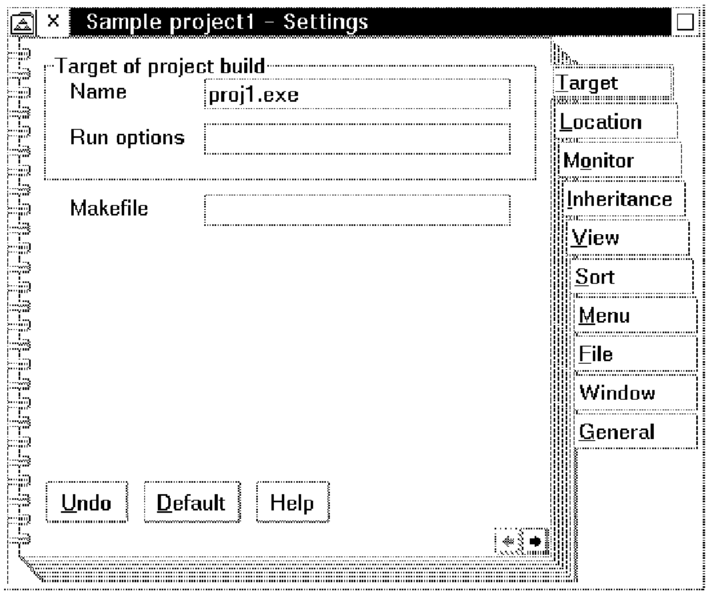
- Specify the directories to be used for the project in the Source directories for projects files list field on the Location page.
- Specify one of the source directories from the list field as the Working directory on the Location page.

Settings – Monitor page

- Set the appropriate monitor settings on the Monitor page.

Settings – Inheritance page

- Add projects from which your project can inherit on the Inheritance page.

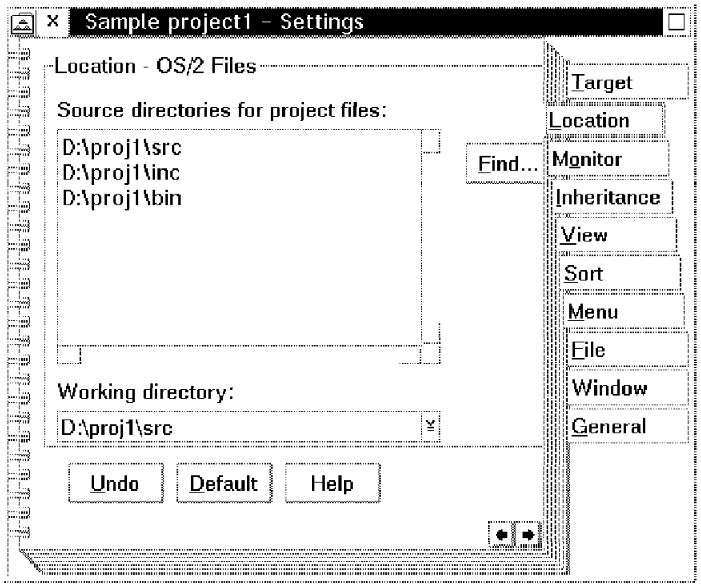


Target of project build

Name: proj1.exe

Run options:

Makefile:



Location - OS/2 Files

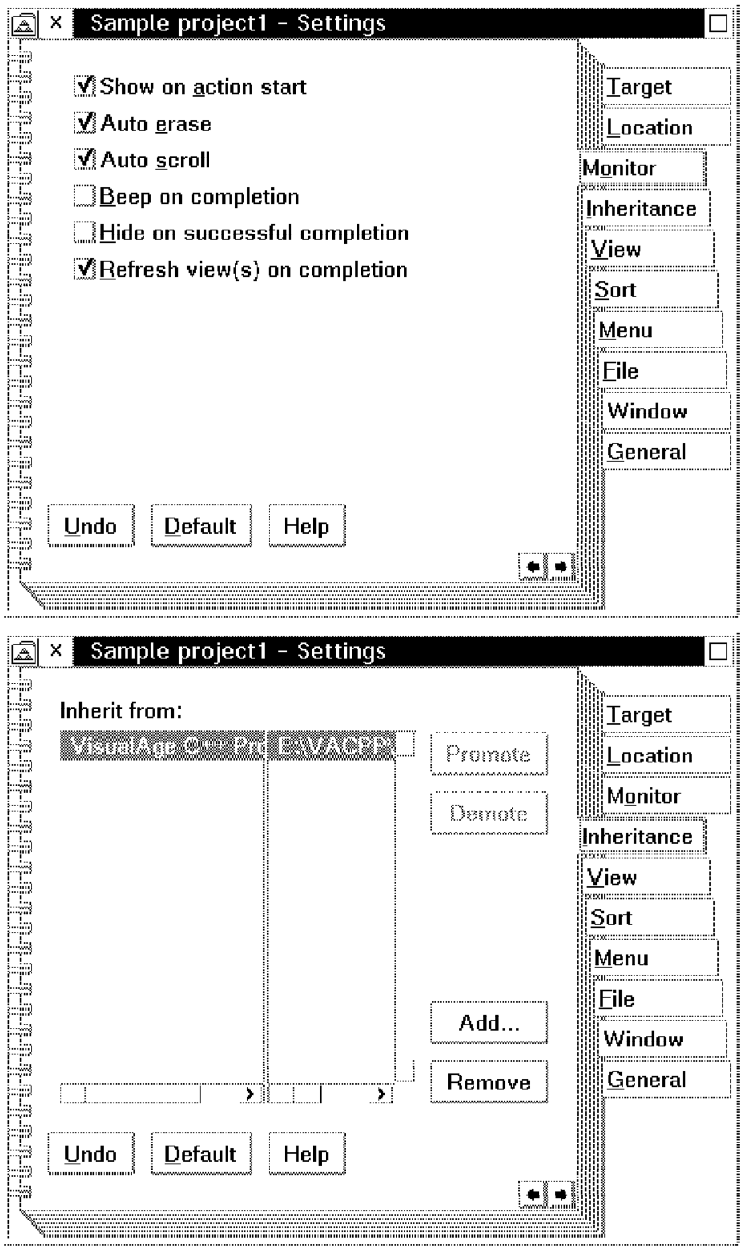
Source directories for project files:

D:\proj1\src
D:\proj1\inc
D:\proj1\bin

Working directory:

D:\proj1\src

Figure 25 (Part 1 of 2). Changing the Project Settings


Sample project1 - Settings

- [✓] Show on action start
- [✓] Auto erase
- [✓] Auto scroll
- [ ] Beep on completion
- [ ] Hide on successful completion
- [✓] Refresh view(s) on completion

Sample project1 - Settings

Inherit from:

VisualAge C++ Pro	E:\VACPP

Figure 25 (Part 2 of 2). Changing the Project Settings

Adding an Action

In this section we show you the steps to add a new action. You begin by selecting the Actions view from the TWI_PRJ - tools setup window.

For each Settings page, we explain the various fields on the page (see Figure 26 on page 71):

Settings – Target page

- Specify the target file name in the Name field.
- Specify the run option for the target file in the Run options field.
- Specify the make file name in the Makefile field.

Add Action – Types page

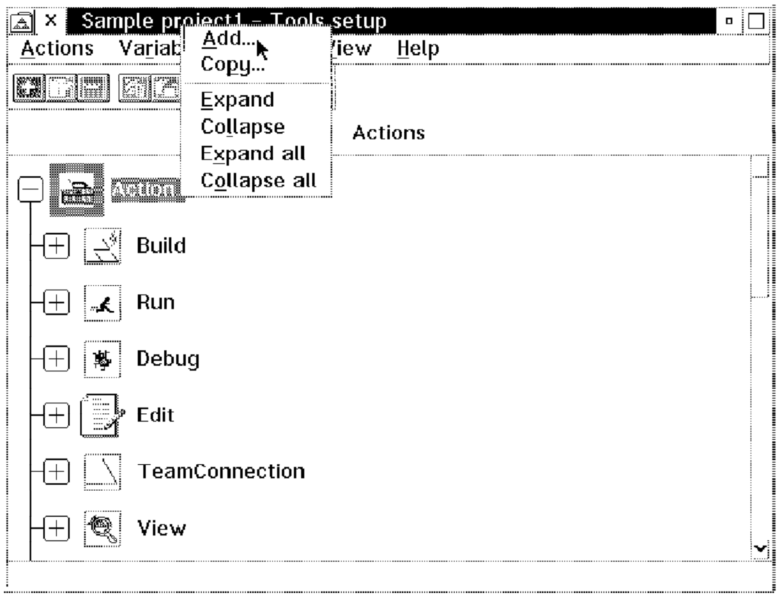
- In the Source types field, specify the types of files that can be applied to the action.
- In the Available types field, you have a list of types defined in the Types view of the tools setup.
- In the Target types field, specify the types of output files required for the various actions.

Add Action – Support page

- Specify the name of the action support DLL in the Action Support DLL – Name field. The action support DLL:
  - Sets the default option
  - Displays the interface for options input
  - Generates the command line to invoke the action command
  - Processes the target and dependencies list
  - Parses selected error messages from the monitor
  - Enables the DDE communication
- Select the priority of the action from the Priority selection scroll list. Actions with higher priority will be displayed at higher positions in the selection scroll list. The default action will be the action with the highest priority.

Add Action – Menus page

- Select *Add to menus*, to display the action in the selected pull-down menu.
- Select *Add to project Options menu* to display the action in the options pull-down menu.
- Select *Add to project toolbar* to display the action in the toolbar.
- Specify in the *Ctrl+Shift+* field a character to be used as an accelerator key.



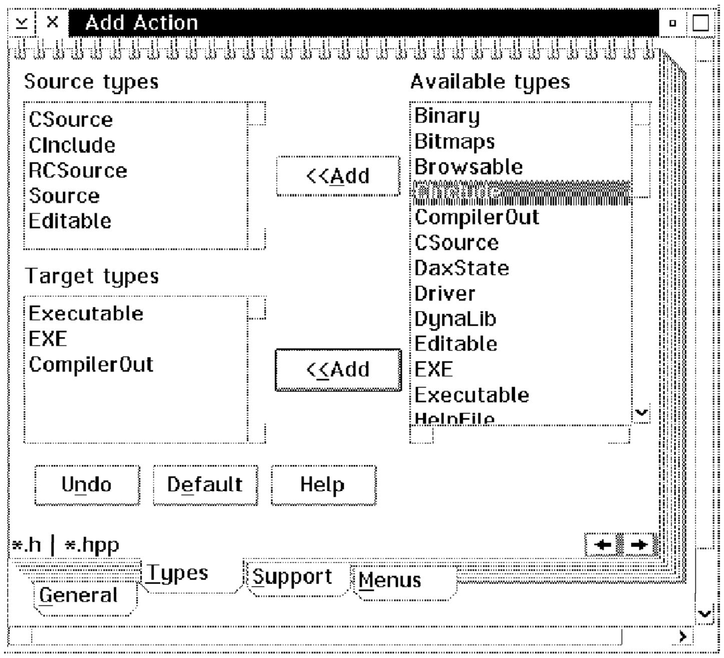
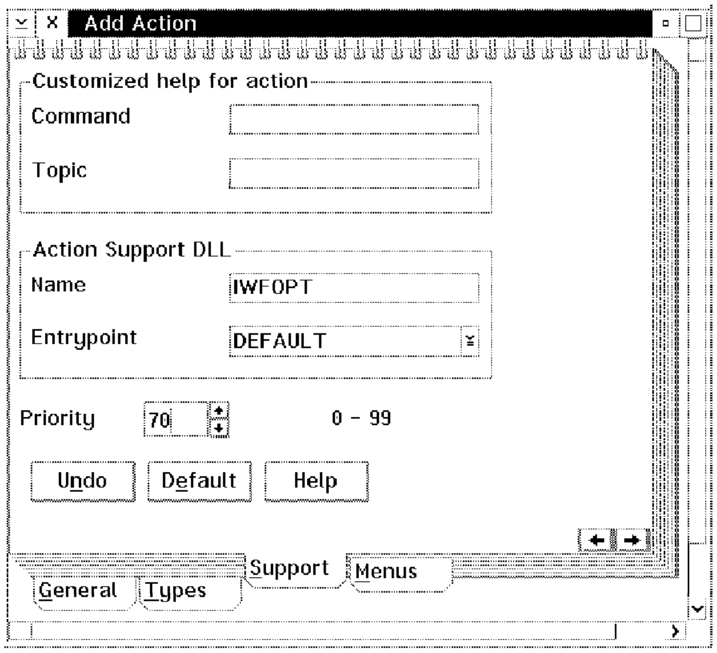


Figure 26 (Part 1 of 2). Adding an Action


Add Action

Customized help for action

Command

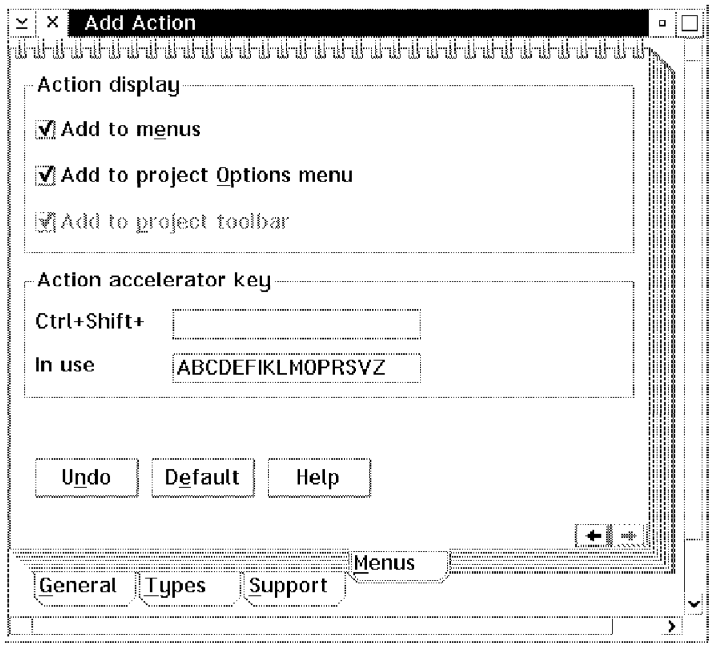
Topic

Action Support DLL

Name: IWFOPT

Entrypoint: DEFAULT

Priority: 70 0 - 99


Add Action

Action display

☑ Add to menus

☑ Add to project Options menu

☑ Add to project toolbar

Action accelerator key

Ctrl+Shift+

In use: ABCDEFIKLMOPRSVZ

Figure 26 (Part 2 of 2). Adding an Action

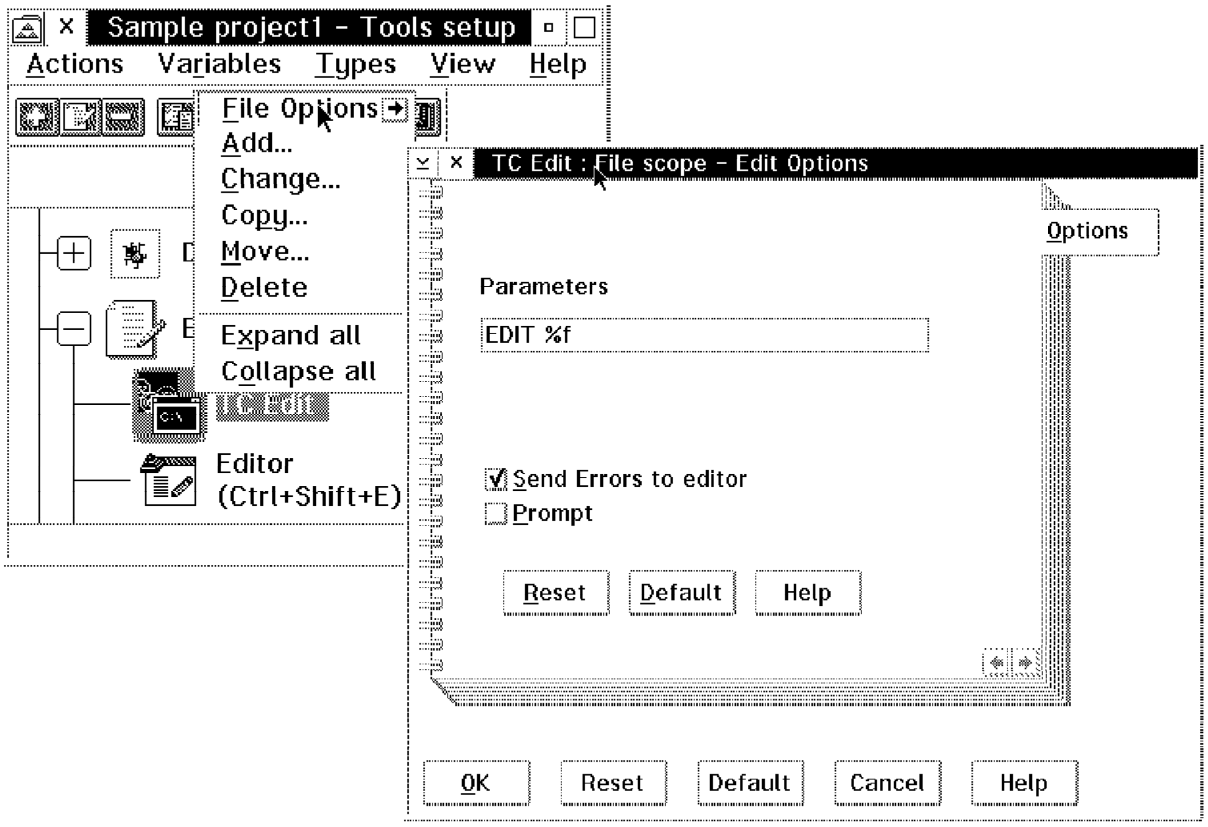
After you have added an action, you can also specify options for the command file invoked from the action. Figure 27 on page 73 shows the options windows you will see if you used *IWFOPT* as the support DLL:

- Specify the parameters for the action (when applied to files) in the *Parameters* field. Use substitution variables to refer to variable items such as file names.

```
%a %z  — specifies all selected files, separated by a space
%f     — specifies the fully qualified name of the first selected file
```

- Select *Send Errors to editor* to enable DDE communication with the editor.
- Specify the parameters for the action (when applied to a project) in the *Parameters* field.
  Substitution variables are different from file options.

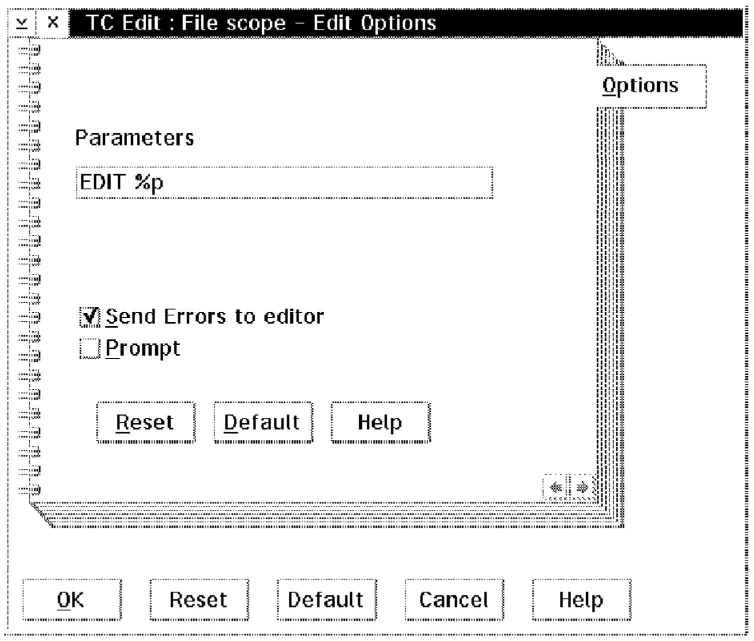
```
%p     — specifies the fully qualified name of the project target
```



Parameters: EDIT %f

[✓] Send Errors to editor
[ ] Prompt

Figure 27 (Part 1 of 2). Changing an Option of an Action



Parameters

EDIT %p

[✓] Send Errors to editor

[ ] Prompt

Figure 27 (Part 2 of 2). Changing an Option of an Action

Actions Defined in the TWI_PRJ Project

Consider the following wish list items:

We expect that:

- Library functions can be executed on WorkFrame parts and projects.
- A tight edit-compile-debug loop will be supported
- The latest code version is always in the library.
- A build should always be done with the library build setup.

In this section we describe the actions that are set in the TWI_PRJ project. For each action we briefly describe its function and provide the following information:

Advantages Explains why we added the function. All of our actions were created to reduce user involvement as much as possible.

Input Shows the input needed for the action other than the environment variables. You will see the input defined in the options of the actions.

Scope Actions are defined as file-scoped, project-scoped, or both.

File-scoped actions are actions that apply to specific files in WorkFrame, and one or more files must be selected to invoke the action. File-scoped actions will show up in the Selected pull-down menu and file pop-up menu.

Project-scoped actions are invoked only on a project, regardless of whether or not a file is selected. Project-scoped actions will show up in the Project pull-down menu and project pop-up menu.

File types Shows the types of files to which the action is applied. For example, the TC Edit action will only be applied to editable files. Types are defined only for file-scoped actions. You will see the types defined in the Types menu of the Tools setup window.

Implementation Shows how we implemented the action.

TC Edit

This action checks out a part, starts an editor, and checks in the selected part. It uses the editor defined in the TC_EDITOR variable.

Advantage Releases users from checking out and checking in a part each time they try to edit it.

Input	Single file name (if using LXSYNC.CMD)
	List of file names (if using EPM.EXE)
Scope	File-scoped
File type	Editable files

Implementation

- Check out the part selected.
- Start an editor (defined in TC_EDITOR) with the selected part name.
- When the edit session is closed, you are asked whether the part should be checked in or not.
- When checkin is selected, invoke the TC CheckIn Forced action.
- If error occurs, abend the action and check the pop-up message.

Use synchronous editor

Although the LPEX editor comes with WorkFrame, you must customize it for TC Edit because it is not a synchronous editor. It stays in memory even when the edit session is closed. We created a command file called **LXSYNC.CMD** to invoke LPEX synchronously.

TC CheckIn

This action checks in one or more parts that you select.

Input	List of file names
Scope	File-scoped
File type	Editable

Implementation

Check in the selected parts, using the TeamConnection command line interface.

TC CheckIn Forced

This action checks in one or more parts that you select. If a part was not checked out previously, it locks the part and checks it in again.

Advantage	You do not have to check out a part.
Input	List of file names
Scope	File-scoped
File type	Editable

Implementation

- Check in the selected parts into TeamConnection.
- If the checkin fails, check the error code.
- If the error code indicates that the part was not checked out, lock the part and try to check it in again.

TC CheckOut

This action checks out one or more parts that you select.

Input	List of file names
Scope	File-scoped
File Type	Editable
Implementation	

Check out the selected parts, using the TeamConnection command line interface.

TC Lock

This action locks one or more parts that you select.

Input	List of file names
Scope	File-scoped
File type	Editable
Implementation	

Lock the selected parts, using the TeamConnection command line interface.

TC UnLock

This action unlocks one or more parts that you select.

Input	List of file names
Scope	File-scoped
File type	Editable
Implementation	

Unlock the selected parts, using the TeamConnection command line interface.

TC Extract

This action extracts one or more parts that you select.

Input	List of file names
Scope	File-scoped
File type	Any
Implementation	

Extract the selected parts, using the TeamConnection command line interface.

TC Touch

This action invokes the Touch part command for one or more parts.

Advantages	Lets you rebuild the target without changing the parts because the build process treats touched parts as being changed.
Input	List of file names
Scope	File-scoped
File type	Any
Implementation	

Invoke the Touch part command for the selected parts, using the TeamConnection command line interface.

TC Build Part

This action invokes TeamConnection build from WorkFrame. The build process runs for one selected file.

Advantages	Because the action is run on a monitor session, you can see the build messages in the WorkFrame monitor. WorkFrame handles the DDE communication to let you jump from the error message in the monitor to the line of a source code where the error occurred. Also, you can start the build process not only from parts with a builder associated with it but also from parts without a builder; that is, you can start the build from a source file or an include file.
Input	List of file names
Scope	File-scoped
File type	Any
Implementation	

- In the WorkFrame tools setup, select Monitor as the session for the action.
- Check whether the selected part has a builder associated with it or not.
- If the part does not have a builder, search for its parent part in the build tree.
- Check whether the parent part has a builder associated with it or not. If it does not, search for the next parent until a part with a builder associated with it is found.

- Execute TeamConnection build with the selected part or the parent parts, using the TeamConnection command line interface.
- Issue the viewmsg command to show the result of the build.

Note: If you double-click on the error message in the monitor, the TC Edit Part action will be invoked, and the part will be checked out automatically.

Why do we have to search for the parent part?

If a part does not exist as a file, you will not be able to start the build process for it. For example, assume that you want to compile source code for the first time. To start a build process you must specify the name of the part that is going to be built. In TeamConnection you trigger the build event for the *target* part of the build, not for the *source* part, as you would normally do for a WorkFrame compile. Since this is the first time you are compiling the code, WorkFrame does not show any file that has the name of the object code. And, you cannot extract the object part from TeamConnection to show it in WorkFrame because the part does not contain anything. Therefore you will not be able to specify the file name to be built in a WorkFrame environment unless you type in the name of the object part.

We implemented a function that searches for the *target* part, a part that has a builder associated with it, from the *source* part that was selected using the build tree. You can select a source code or an include file and start the build process of the object part. There may be two or more object parts, but we build only the first object part that is found.

TC Build Part Forced

This action invokes the TeamConnection build process with force option.

Advantage	You can rebuild everything regardless of the state of the last build.
Scope	File-scoped
File type	Any

TC Build Target

This action invokes a TeamConnection build for the target file specified in the TC_TARGET_PART environment variable.

Advantage	You can build the whole build tree for the build target. This enables you to build the whole application or project.
Scope	Project-scoped

TC Build Target Forced

This action invokes the TC Target Build action with the force option.

Scope Project-scoped

Show WorkArea

This action invokes a TeamConnection GUI session that shows the Partfull view of the work area.

Scope Project-scoped
Implementation

Invoke the Partfull view, using the TEAMCGUI command.

Note: Our intention was to show the part view of the TeamConnection client. However, TeamConnection does not provide an official interface for the TEAMCGUI command, so we could only invoke the Partfull view. We found the TEAMCGUI options by using the integrated interface that TeamConnection provides.

Freeze WorkArea

This action freezes the work area to make a backup of a whole project so that you can return to the same stage of development.

Scope Project-scoped
Implementation

Invoke the freeze workarea command, using the TeamConnection command line interface.

TC Task List

This action invokes the Task List window of the TeamConnection GUI. You can switch to the TeamConnection GUI whenever you have to.

Scope Project-scoped
Implementation

Invoke the Task List window of the TeamConnection client, using the TEAMCGUI command.

TC View Build Messages

This action shows the latest build message, sent to the WorkFrame monitor, of the selected part.

Scope File-scoped

Implementation

Invoke the Touch part command for the selected part(s), using the TeamConnection command line interface.

Chapter 6. Using WorkFrame Projects

In this chapter we describe a way of importing an existing WorkFrame project into a TeamConnection project. We believe that this will give directions of how to migrate your existing WorkFrame projects to a TeamConnection environment.

Importing Existing WorkFrame Projects

Consider the following wish list item:

We expect that:

- A WorkFrame project can be automatically imported to a TeamConnection work area.

Basically WorkFrame files (that is, source code, object code, and executable code) are implemented as TeamConnection parts. And WorkFrame build descriptions in a make file are converted to a TeamConnection build tree. To create a new part in a TeamConnection environment, you have to create other TeamConnection objects in advance.

The fundamental steps for migration are:

1. Create a TeamConnection family
2. Create a component structure
3. Create a release and a work area
4. Create parsers and builders
5. Determine the import rules
6. Convert a make file into a command file

TeamConnection provides a utility called FHOMIGMK that will convert an NMAKE make file into a TeamConnection command file. But we will describe this step in general to help you import from other make files also.

7. Run the command file

8. Verify the results

TeamConnection Is Case Sensitive!

When importing WorkFrame projects to TeamConnection, remember that TeamConnection is case sensitive. You cannot use a different case for the same object. For example, if an include file named *pgm01.c* is referred to as *pgm01.C* in a source file, you must either rename the include file name or modify the source code.

Also be careful of the file system you are using. If you are using a file allocation table (FAT) file system, all file names are in upper case. If you refer to those files in a make file in lower case, TeamConnection will not be able to find the file during import. To avoid this, set TC_CASESENSE=LOWER or change the TeamConnection settings to lower case.

Note: TC_CASESENSE is valid only for TeamConnection commands. It is not valid for build actions or reports.

Creating a TeamConnection Family

We recommend that you use a new family to import WorkFrame projects. If you are using an existing family, do not forget to take a backup before migration because it may be extremely difficult to roll back an import.

Creating a Component Structure

When you import a WorkFrame project into the TeamConnection environment, you must create a component structure or at least one component to attach the WorkFrame files. You must consider access control, problem tracking, and organizational structure to have the most suitable component structure. Keep the component structure as simple as possible so that the parts can be imported automatically. Refer to “Grouping the Parts and Controlling Access” on page 16 for more information about assigning parts to components.

Assigning a Part to a Component

In “Grouping the Parts and Controlling Access” on page 16 we discussed one way of assigning parts to components. The following are some other ways of assigning a part to a component:

- Attach all parts from WorkFrame projects to one component and then distribute to final components later on.

  You will be able to check whether all parts have been assigned to the correct component. If they have not, you will see that in the first component. The disadvantage of this approach is that the distribution of the parts will be lengthy.

  Of course you do not have to distribute the parts if you have decided to use only one component in your component structure. But you will not be able to control access to the parts between the projects.

- Attach parts from one project to different components.

  This approach will be sufficient if you want to control access to some files within a project. For example, you can attach common parts to a different component to give read-only access to developers. When you are taking this approach you must know exactly which parts are attached to each component. You need a strict naming convention to distinguish common parts from local parts.

Creating a Release and a Work Area

In this step you create a release and a work area, using your naming convention. This step is a prerequisite for creating parsers and builders.

Creating Parsers and Builders

You do not have to create a parser or a builder until you run a build process in TeamConnection. If you are considering creating a build tree from a make file, however, you should create parsers and builders at this point. It is your responsibility to create parsers and builders from a make file definition or other definitions.

Parser
The function of a parser is to analyze source files and parse all dependent files.

Builder
For each compiler and each linker you should create a builder. Do not use the compiler itself as a build script because the build script is copied to the cache directory for performance. Also, the compiler will stay in memory, so you may get unexpected results on the next build.

Sample parsers are available for C (fhbopars.cmd), COBOL (fhbcbprs.cmd), and PL/I (fhbplprs.cmd).

TeamConnection also provides sample build scripts for the VisualAge C++ compiler (fhbocomp.cmd); VisualAge C++ link (fhbolin2.cmd); VisualAge C++ ilink (fhbolink.cmd); VisualAge COBOL compiler (fhbcob2.cmd); VisualAge COBOL link (fhbcob2l.cmd); Resource Compiler (fhborc.cmd); OS/2 PL/I compiler (fhbplbld.cmd); and OS/2 PL/I link (fhbpllnk.cmd). We recommend that you create different builders when using the same compiler with different compiler options. For example, you should create a builder for production and a builder for debugging.

For more information about parsers and builders, see the *IBM TeamConnection for OS/2 User's Guide*, SC34-4499.

Note: The FHOMIGMK import utility uses default TeamConnection parser names and TeamConnection builder names. The default parser for C or C++ source code is called **c**, and for resource files it is called **rc**. The default builder for the C and C++ compiler is **icc**, and for the C and C++ linker it is **linker**. Other default builders are **ipfc** for the help file compiler and **res** for the resource compiler. Create these parsers and builders for the release before you run this utility.

Determining the Import Rules

When you are importing make files with the FHOMIGMK utility, you have to prepare a file that contains all rules for importing the make files. This rules file is also useful for checking whether you have defined all information required to import a WorkFrame project, even if you are not using the FHOMIGMK utility. If you create a rules file, it will be easier to import a project that does not use make files. It may be easier to modify a default rules file that TeamConnection provides than to create a new one. The default rules file is TEAMC\BIN\FHOMIGMK.RUL

In this section we create a new rules file to import the sample WorkFrame projects shown in Figure 28 on page 87. Each sample project has a root directory and some subdirectories containing different types of files.

```
project1    D:\proj1\bin  - contains target files   (*.exe, *.obj, *.res)
            D:\proj1\src  - contains source files   (*.c)
                            contains resource files (*.rc)
                            contains make files     (*.mak)
                            contains map files      (*.map)
                            contains icon files     (*.ico)
            D:\proj1\inc  - contains include files  (*.h)

project2    D:\proj2\bin  - contains target files   (*.exe, *.obj, *.res)
                            contains map files      (*.map)
            D:\proj2\src  - contains source files   (*.c)
                            contains resource files (*.rc)
                            contains make files     (*.mak)
                            contains ipf files      (*.ipf)
            D:\proj2\hlp  - contains icon files     (*.ico)
                            contains help files     (*.hlp)
            D:\proj2\inc  - contains include files  (*.h)
```

Figure 28. Sample WorkFrame Projects

1. Determine the component

   First of all you must determine the components to which parts are being attached. For each part, define a component. To make life easier, use wild cards (* and ?) in part names.

   Generally WorkFrame projects have a directory structure for project management. If so, you can use this directory structure as a file mask.

   For example, let us assume that project1 in Figure 28 will be attached to a component, *server*, and project2, to a component, *client*. Your rules file will contain file masks and components as in Figure 29.

```
  file masks         components
---------------------------------
  D:\proj1\*           server
  D:\proj2\*           client
```

Figure 29. Example of a Rules File: Components

2. Define the file types and the connections

   You must decide on the file type of each part. Parts can be either binary, text, or none, depending on the data they contain:

   binary
      Indicates that the part being created is a binary file

text
Indicates that the part being created is a text file (the default)

none
Indicates that the part being created will never contain data. This type will not be used if you are importing an existing WorkFrame project.

You also must decide on the connection between a part and its parent:

input
Specifies that the part is an input to the build of its parent. A source file is an example.

output
Specifies that the part is built at the same time its parent is built. A map file is an example.

dependent
Specifies that the part is needed to build its parent, but it is not an input to the build. In C, #include file is an example.

none
Specifies that the part will not be connected to another part even though a dependency was found in the make file.

Note: In the sample rules file, it is stated that you can use an option called *associate*. Do not use that option because TeamConnection does not accept it.

To define both file type and connection, change or add new file masks in the rules file; that is, create a file mask to distinguish a binary file from a text file and create another mask to distinguish a connection from being either *input*, *output*, or *dependent*. Your rules file would then look like that in Figure 30 on page 89.

```
 file masks          type     connect    components
------------------------------------------------------
 D:\proj1\bin\*.exe   binary   input       server
 D:\proj1\bin\*.obj   binary   input       server
 D:\proj1\bin\*.res   binary   input       server
 D:\proj1\src\*.map   text     output      server
 D:\proj1\src\*.c     text     input       server
 D:\proj1\src\*.ico   binary   input       server
 D:\proj1\src\*.rc    text     input       server
 D:\proj1\inc\*.h     text     dependent   server

 D:\proj2\bin\*.exe   binary   input       client
 D:\proj1\bin\*.obj   binary   input       server
 D:\proj1\bin\*.res   binary   input       server
 D:\proj2\bin\*.map   text     output      client
 D:\proj2\src\*.c     text     input       client
 D:\proj2\src\*.rc    text     input       client
 D:\proj2\src\*.ipf   text     input       client
 D:\proj2\hlp\*.ico   binary   input       client
 D:\proj2\hlp\*.hlp   binary   input       client
 D:\proj2\inc\*.h     text     dependent   client
```

Figure 30. Example of a Rules File: File Types and Connections

3. Define the initial contents and associate the builders and the parsers

   Define the path and the file name from which a part acquires its contents. The initial content will then be imported into a TeamConnection database.

   The parameters for initial contents are:

   ***** Contents are expected to be in the directory specified in a make file.

   none
   Specifies that this part does not have contents. Examples would be target files and output files.

   directory name
   Contents are expected to be found under the directory with the name specified in a make file. For example, if a file name, src\xxx.c, is found in a make file, and the contents are d:\ppp, TeamConnection looks for d:\ppp\src\xxx.c.

   As stated before, you should define parsers and builders before this step. Associate the parsers and builders with their respective parts.

   Also define parameters for the builders if there is any. Parameters should be enclosed by double quotations if there are spaces in them.

   By including these information, your rules file would be as in Figure 31 on page 90.

```
  file masks           type     builder  parser  connect    content  parameter  component
-------------------------------------------------------------------------------------------
  D:\proj1\bin\*.exe   binary   linker   none    input      none     none       server
  D:\proj1\bin\*.obj   binary   icc      none    input      none     none       server
  D:\proj1\bin\*.res   binary   res      none    input      none     none       server
  D:\proj1\src\*.map   text     none     none    output     none     none       server
  D:\proj1\src\*.c     text     none     c       input      *        none       server
  D:\proj1\src\*.ico   binary   none     none    input      *        none       server
  D:\proj1\src\*.rc    text     none     rc      input      *        none       server
  D:\proj1\inc\*.h     text     none     c       dependent  *        none       server

  D:\proj2\bin\*.exe   binary   linker   none    input      none     none       client
  D:\proj2\bin\*.obj   binary   icc      none    input      none     none       client
  D:\proj2\bin\*.map   text     none     none    output     none     none       client
  D:\proj2\bin\*.res   binary   res      none    input      none     none       server
  D:\proj2\src\*.c     text     none     c       input      *        none       client
  D:\proj2\src\*.rc    text     none     rc      input      *        none       client
  D:\proj2\src\*.ipf   text     none     ipf     input      *        none       client
  D:\proj2\hlp\*.ico   binary   none     none    input      *        none       client
  D:\proj2\hlp\*.hlp   binary   ipfc     c       input      none     none       client
  D:\proj2\inc\*.h     text     none     c       dependent  *        none       client
```

Figure 31. Example of a Rules File 3

Convert a Make File into a Command File

The TeamConnection *FHOMIGMK* import utility translates WorkFrame make files into TeamConnection command files, using the rules file.

Here is the basic action of FHOMIGMK:

1. FHOMIGMK runs the nmake command with the **/p** option.

   The **/p** option writes out all macro definitions and target definitions from the make file. Figure 32 shows a sample make file.

```
.SUFFIXES: .cpp .obj .rc .res

.rc.res:
    @echo ″ Compile::Resource Compiler ″
    rc.exe -r %s d:\proj1\bin\%|fF.RES

.cpp.obj:
    @echo ″ Compile::C++ Compiler ″
    icc.exe /Id:\proj1\inc; /Fo″D:\proj1\bin\%|fF.obj″ /C %s

\proj1\bin\proj1.exe: \
    \proj1\bin\proj1.obj \
    \proj1\bin\proj1.res
    @echo ″ Link::Linker ″
    @echo ″ Bind::Resource Bind ″
    icc.exe @<<
     /Fed:\proj1\bin\proj1.exe
<<
    rc.exe \proj1\bin\proj1.res \proj1\bin\proj1.exe

\proj1\bin\proj1.res: \
    \proj1\src\proj1.rc \
    \proj1\inc\proj1.ico \
    \proj1\inc\proj1.hpp

\proj1\bin\proj1.obj: \
    \proj1\src\proj1.cpp \
    \proj1\inc\proj1.hpp \
```

Figure 32. Sample Make File for Project1

2. FHOMIGMK extracts file names and dependencies

   Running nmake with the **/p** option creates output in the format shown in Figure 33 on page 92.

```
MACROS:

       INCLUDE = D:\IBMCPP\INCLUDE;D:\IBMCPP\INCLUDE\OS2;D:\IBMCPP\INC;D:\IBMCPP\INCLUDE\SOM
WORKPLACE__PROCESS = NO
        VBPATH = .;D:\IBMCPP\DDE4VB
            BC = bc
            CC = icc

              •
 INFERENCE RULE:


 .rc.res:

      commands:        @echo " Compile::Resource Compiler "
       rc.exe -r %s d:\proj1\bin\%|fF.RES

 .cpp.obj:

      commands:        @echo " Compile::C++ Compiler "
       icc.exe /Id:\proj1\inc; /Fo"D:\proj1\bin\%|fF.obj" /C %s


 TARGETS:

 \proj1\bin\proj1.obj:

      flags:
      dependents:     \proj1\src\proj1.cpp
                      \proj1\inc\proj1.hpp
      commands:

 \proj1\bin\proj1.exe:
      flags:
      dependents:     proj1.obj proj1.res
      commands:        icc.exe @<<
      /Fed:\proj1\bin\proj1.exe
       \proj1\bin\proj1.obj
 <<
       rc.exe d:\proj1\bin\proj1.res d:\proj1\bin\proj1.exe

 \proj1\bin\proj1.res:

      flags:
      dependents:     d:\proj1\src\proj1.rc
                      d:\proj1\inc\proj1.ico
                      d:\proj1\inc\proj1.hpp
      commands:
```

Figure 33. Sample Output from nmake

FHOMIGMK creates parts from the *TARGETS:* field and the *dependents:* field. For each target, a *create part* statement is generated, and for each dependent, a *create part* statement and a *connect* part statement are generated.

The Name of a Part

Set a path name in the TC_MAKEIMPORTTOP variable to strip off the leading part of the path name from the part name.

For example, assume you have TC_MAKEIMPORTTOP set to the path *D:\proj1*. If your make file references file *proj1.obj* in directory *d:\proj1\bin*, FHOMIGMK will use *bin\proj1.obj* as a part name.

3. FHOMIGMK checks the rules file and sets options for each statement created in step 2.

It is advisable to run FHOMIGMK with the **/k** option. This option creates a command file without executing it. This enables you to check the command file before creating anything in TeamConnection.

Figure 34 shows a sample command file created by FHOMIGMK.

Note: In this example, the family, release, and work area options are stripped off for ease of understanding. You can set these options in two ways: (1) Set them as FHOMIGMK utility options, or (2) set the TC_FAMILY, TC_RELEASE, and TC_WORKAREA environment variables for each option.

```
   teamc part -create  proj1\bin\proj1.exe -binary  -builder linker -empty  -component server
   teamc part -create  proj1\bin\proj1.obj -binary  -builder icc    -empty  -component server
   teamc part -create  proj1\bin\proj1.res -binary  -builder res    -empty  -component server
   teamc part -create  proj1\src\proj1.cpp -text    -parser c       -from d:\proj1\src\proj1.cpp -component se
   teamc part -create  proj1\src\proj1.rc  -text    -parser rc      -from d:\proj1\src\proj1.rc  -component se
   teamc part -create  proj1\src\proj1.ico -binary                  -from d:\proj1\inc\proj1.ico -component se
   teamc part -create  proj1\inc\proj1.hpp -text    -parser c       -from d:\proj1\inc\proj1.hpp -component se
   teamc part -connect proj1\src\proj1.ico -parent proj1\bin\proj1.res -input
   teamc part -connect proj1\src\proj1.rc  -parent proj1\bin\proj1.res -input
   teamc part -connect proj1\inc\proj1.hpp -parent proj1\bin\proj1.res -dependent
   teamc part -connect proj1\src\proj1.cpp -parent proj1\bin\proj1.obj -input
   teamc part -connect proj1\inc\proj1.hpp -parent proj1\bin\proj1.obj -dependent
   teamc part -connect proj1\bin\proj1.obj -parent proj1\bin\proj1.exe -input
   teamc part -connect proj1\bin\proj1.res -parent proj1\bin\proj1.exe -input
```

Figure 34. Sample Command File for project1

Unsupported Syntax

FHOMIGMK does not support the full syntax in a make file dependency list:

- Paths enclosed by {}

  With nmake you can use {} to search for dependent files; for example:

  ```
  targets : {directory1,directory2...}dependents
  ```

  But FHOMIGMK will try to create a part named {directory1,directory2...}dependents. To prevent this, change {} to the actual path name instead.

- Same target in several description blocks

  With nmake you can use a file in more than one description block; for example:

  ```
  target :: dependent1
         command1
  target :: dependent2
         command2
  ```

  FHOMIGMK will try to create two parts, *target* and *target* :. To prevent this, split the make file in two and create a different build tree for each make file.

If You Do Not Have a Make File...

If you do not have a make file that represents your WorkFrame project, you can use the WorkFrame makemake utility to create one, or you can try the approach presented here. We assume that you have taken all of the fundamental steps for migration described in this chapter, up to creating a rules file (that is, steps 1 through 4).

1. Create a make file with a basic dependency list.

   A basic dependency list is a dependency list without the include files. The dependencies of the source code must be defined in your make file. So your make file has all information except the include file dependencies.

2. Import the make file, using the FHOMIGMK utility.

   In this step you create a basic build tree in TeamConnection. Before running the utility, check that all parsers are correctly defined in the rules file. Your build tree will have the following structure:

   ```
   ─ EXE
      │
      ├─ OBJ ── SOURCE
      │
      ├─ OBJ ── SOURCE
      │
      └─ OBJ ── SOURCE
   ```

3. Create parts for all include files.

   Use the TeamConnection command line interface for this step rather than using a WorkFrame action provided by TeamConnection. You will need one TeamConnection command statement for each include file, as in this example:

   ```
   teamc -part create part name -from include file name
         -component server -parser c
   ```

4. Run build in TeamConnection

   When you run the build function, the parsers get all of the dependencies of the source parts and create a build tree with all the dependency parts in it.

You now have a complete build tree with all dependencies connected to the parent parts.

Verifying the Results

Always verify the results of running the FHOMIGMK utility because they may not be what you expected. If the results are not what you expected, check each migration step to solve the problem. The build tree can always be corrected by using the **part -connect** and **part -disconnect** commands.

For verification, build the target file, using the TeamConnection build function and compare it with the target file created in WorkFrame.

Appendix A. Sample REXX Programs

We do not know whether there ever will be a general and automated approach to integrating WorkFrame and TeamConnection that will enable you to maintain the full flexibility of both products. But with the current versions, you have to devise *your* own integration approach, if you want to use the two products together in a workable way. This implies that you will have to spend some time defining the way your developers should work with the tools and provide a set of WorkFrame actions, variables, and types that best meets your needs.

The REXX programs included in this appendix provide an example of integrating the two products. Unfortunately, the TeamConnection integration approach implemented by TWI_DO as well as the automatic generation of a new project (TWI_PRJ, TWI_XBT,TWI_GPT) using the Project Smarts interface of WorkFrame suffer from some restrictions in the current levels of WorkFrame and TeamConnection.

Disclaimer.

Do not expect a complete solution from the REXX programs included in this appendix. The programs were written in only a few weeks to evaluate the feasibility of a customized WorkFrame/TeamConnection integration setup. They are only meant to serve as an example or a base from which you can begin to develop your own setup!

For the purpose of readability, we group the code in a logical way. Therefore some pages have more white space than others.

Notes on Project Smarts

In this section we present some notes and remarks from our findings in trying to integrate the TeamConnection and WorkFrame products.

Documentation

The Project Smarts section of the *VisualAge for C++ User′s Guide and Reference* contains the following discrepancies:

The NewLine character in a REXX string is not ′\n′. You can specify a NewLine in a REXX string in several ways. The following is one way of achieving readability:

```
   ...
 ″PAMLOCATION:IWFBPAM=″path1 || NewLine(),
                     || path2 || NewLine(),
                        ...
```

where `NewLine` is defined as:

```
...
NewLine:
 Return ′0a′x
...
```

The backslash (\) is handled as a control character by some of the Project Smarts functions, such as:

- IwfQueryVariables
- IwfSaveVariables

If, for example, you provide the `c:\prj\build` default path before calling *IwfQueryVariables()*, the path will show up as something like `c:\prj◘uild`, because the C program recognizes the `\b` as a backspace control character.

The meaning of return codes like 25xxx can sometimes be determined by subtracting 25000 from the return code and issuing HELP xxx on the OS/2 command line. This of course only works if the WorkFrame code uses the “standard” OS/2 error numbering scheme.

Return code 25002 seems to mean *project file not found*. If, for example, you want to use the *IwfAddVariable()* function to add some variables to a project with a name containing *NewLine* characters, you will get this error if you use the project′s name as the file name portion in the `stem.pszProject` parameter (path of project file).

In this case, use the value returned in the `stem.pszTargetFile` variable by the *IwfCreateProjectFrom...()* functions!

Return Code 25013 seems to mean *variable or action already defined*.

SysCreateObject(IWFProject,...)

When spanning the call to SysCreateObject over several lines, remember that the different parameters for this function have to be separated by commas, and that the line continuation character of REXX is a comma too. So a call to *SysCreateObject()* has to look like the following:

```
Call SysCreateObject(″IWFProject″ ,,
                     my_title ,,
                     ... ,,
                     setup_string ,,
                     options);
```

or:

```
Call SysCreateObject(″IWFProject″ ,
                    ,my_title ,
                    ,... ,
                    ,setup_string ,
                    ,options);
```

Therefore:

- Do not use spaces between *SysCreateObject()* and the ″(″.
- Use two subsequent commas (,) if you want to split a line with a new parameter.

TRUE or FALSE does not work as values for the MON... controls in the Object Setup string. Unfortunately, the values accepted for the

- MONAUTOSCROLL
- MONAUTOERASE
- MONDISPLAYONSTART
- MONHIDEONCOMPLETION

controls are not TRUE and FALSE as stated in the *VisualAge for C++ User′s Guide and Reference*, but **YES** and **NO** as shown in this example:

```
setup_string = ″RUNPROMPT=FALSE;″ ,
            || ″RUNMONITORED=FALSE;″ ,
               ...
            || ″MONAUTOSCROLL=YES;″ ,
            || ″MONAUTOERASE=NO;″
```

IwfCreateProjectFromProject

Undocumented return parameter stem.pszTargetFile
Contrary to the documentation, *all* of the *IwfCreateProjectFrom...()* functions, seem to return a file name of the project file that is different from the title specified in `stem.pszTargetProject`, if for example, NewLine characters are used in the title.

IwfQueryVariables

Carriage Return - Line Feed (CrLf) sequences may be returned in the `stem.pszVariableValue.i` strings if you press `Enter` in the Variable Setting field of the dialog window. A simple way to get rid of these *CrLf* sequences is to `Translate` them to blanks, using the following code sequence:

```
stem.pszVariableValue.i = Translate(stem.pszVariableValue.i,"  ",CrLf())
```

where `CrLf` is defined as:

```
CrLf:
 Return '0d0a'x
```

IwfSaveVariables and IwfRestoreVariables

If the value of a variable has been set to the null string in the *IwfQueryVariables* dialog window and you are using the *IwfSaveVariables()* and *IwfRestoreVariables()* functions to save and restore the values for the next invocation, be aware that, after running *IwfRestoreVariables()*, the value of a `stem.pszVariableValue.i` variable will no longer be the null string, but the contents of `stem.pszVariableName.i` (the name of the variable surrounded by % characters).

IwfAddVariable

If stem.pszValue is the null string when IwfAddVariable is invoked, the value of the variable in the project will actually be set to the value of `stem.pszName` surrounded by % characters.

This might not have the effect you intended. So take appropriate action and set the variable to a single-space character or something appropriate before calling *IwfAddVariable()*.

File and Project Action Options of IWFOPT.DLL

Never specify %a %z twice in the same option string in WorkFrame fix levels below CTW303. There currently exists a problem in WorkFrame that will hang your system when the `%a %z` (all selected files) substitution string is used more than once in the same option string. The problem will be fixed in a later update of WorkFrame.

There is no way to set action options from a REXX program
Unfortunately, the default action support DLL (IWFOPT.DLL) does not provide an interface to set the option strings for file and project actions from a REXX command file. The only way to set or modify the settings is through the GUI.

Defining Our Own Set of WorkFrame Actions

TWI_DO: TeamConnection Actions for WorkFrame

This REXX program implements the TeamConnection actions for WorkFrame. TWI_DO.CMD (see Figure 35 on page 103) basically uses the TeamConnection command line interface as well as the REXX queue (RxQueue) interface to accomplish this task.

The program basically:

- Checks that all required environment information is available

  and
- Combines one or more logical steps into one WorkFrame action (for example, *CheckOut*, *Edit*, and *CheckIn*).

For a list of the available functions please see "Actions Defined in the TWI_PRJ Project" on page 75.

Note: This program provides a set of useful actions to perform TeamConnection functions from the WorkFrame environment. Nevertheless, this is only one possible approach. If your requirements or wishes are different, you should be able to easily enhance or modify this program to fit your needs.

```
/* ********************************************************************
  TeamConnection / WorkFrame integration support macro

  File ..:  TWI_DO.CMD
  Project:  A TeamConnection/WorkFrame integration approach ...

  Author :  L. Sparmann  (SPARMANN @ BOEVM2)
  Owner .:  L. Sparmann  (SPARMANN @ BOEVM2)

  Description :

   This program is implementing a set of TC functions as WF actions...

  Version and Change History:
  --------------------------*/
  version = "1.00 dated:  4.Apr.96  by: L. Sparmann"
  reason  = "Initial version ..."
/*
*********************************************************************/
 Address CMD
"@ECHO OFF"
                                       /* ----------------------------*\
                                         Setup for error trapping
                                       \*----------------------------*/

 Signal on  NOVALUE  ; Signal on  SYNTAX ;           /* SIGNAL only ! */
 Call   on  FAILURE  ; Call   on  ERROR  ;           /* SIGNAL & CALL */
 Call   on  NOTREADY ; Signal on  HALT   ;           /* SIGNAL & CALL */

 /* ------------------------------------------------------------------*\
    Set up the global variables...
 \*------------------------------------------------------------------*/
 XRc = 0

 /* ------------------------------------------------------------------*\
    Basic setup ...
 \*------------------------------------------------------------------*/
 g.            = "?"                  /* All my global variables ... */
 g.exitcmds    = ""                   /* Cmds to be executed on Exit */
```

Figure 35 (Part 1 of 25). TWI_DO.CMD

```
                             /* ---------------------------------------*\
                                Get the call environment
                             \*----------------------------------------*/
Parse Upper Source g.myenv g.mycall g.mypname .
Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
g.mypath  = FileSpec("PATH" ,g.mypname)
g.mydrive = FileSpec("DRIVE",g.mypname)
                             /* ---------------------------------------*\
                                Load OS/2 built-In Utilities
                             \*----------------------------------------*/
If (RxFuncQuery("SysLoadFuncs")) Then Do
  Call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
  Call SysLoadFuncs
  End
                             /* ---------------------------------------*\
                                It's good practice to clean up the
                                 REXX Queue in the beginnig
                             \*----------------------------------------*/
Do Queued(); Pull .; End
                             /* ---------------------------------------*\
                                Get a system unique name
                                 (using the REXX Queue for this)
                             \*----------------------------------------*/
g.myqid = RxQueue("CREATE",)
dyRc    = RxQueue("DELETE",g.myqid)    /* we do not need the Queue ! */
                             /* ---------------------------------------*\
                                Create a unique temporary filename
                             \*----------------------------------------*/
tmp = Value("TMP",,"OS2ENVIRONMENT")
If (tmp = "") Then Do
  tmp = Value("TEMP",,"OS2ENVIRONMENT")
  If (tmp = "") Then Do
    Call Abort(200,
              "There is no TMP or TEMP environment variable specified!")
    End
  End

g.tmpfile = SysTempFileName(Strip(tmp,"T","\")"\"Left(g.myqid,8)".???")
g.exitcmds = g.exitcmds "ERASE" g.tmpfile "2>NUL "
```

Figure 35 (Part 2 of 25). TWI_DO.CMD

```
/* ----------------------------------------------------------------*\
   Look for debug or trace setings
\*-----------------------------------------------------------------*/
If (Value("TWI_TRACE",,"OS2ENVIRONMENT") = "ON") Then g.trace? = 1
                                                 Else g.trace? = 0

/* ----------------------------------------------------------------*\
   Set up local stuff
\*-----------------------------------------------------------------*/
                            /* -------------------------------------*\
                               TC Error Message Codes
                            \*--------------------------------------*/
tc_err_PartDoesNotExist  = "0010-052"
tc_err_PartNotCheckedOut = "0010-264"

/* ----------------------------------------------------------------*\
   Get Arguments (as typed = mixed case) and provide help if desired
\*-----------------------------------------------------------------*/
Parse Arg allargs

If (allargs = "") ,
 | (allargs = "?") Then Do
  Say g.myname":"
  Say
  Say "  Syntax:" g.myname " function  funcparms"
  Say
  Say "Additional output for problem determination will be produced,"
  Say  "if the TWI_TRACE environment variable is set to ON."
  Say
  Exit 100
  End
/* ----------------------------------------------------------------*\
   Ensure, that the TC environment is defined completely, so that
    we can do our job without asking stupid questions ...
\*-----------------------------------------------------------------*/
                            /* -------------------------------------*\
                               Required information ...
                                ... from the TC standard environment
                            \*--------------------------------------*/
XRc = XRc + Get_and_Check_TC_Variable("TC_USER"     )
XRc = XRc + Get_and_Check_TC_Variable("TC_FAMILY"   )
XRc = XRc + Get_and_Check_TC_Variable("TC_RELEASE"  )
XRc = XRc + Get_and_Check_TC_Variable("TC_WORKAREA" )
XRc = XRc + Get_and_Check_TC_Variable("TC_CASESENSE")
XRc = XRc + Get_and_Check_TC_Variable("TC_BUILDPOOL")
```
Figure 35 (Part 3 of 25). TWI_DO.CMD

```
                              /* --------------------------------------*\
                                  ... that is WF integration specific
                              \*---------------------------------------*/
   XRc = XRc + Get_and_Check_TC_Variable("TWI_RELATIVE" )
   XRc = XRc + Get_and_Check_TC_Variable("TWI_EDITOR"   )

  "SET TC_TOP=" /* If TC_TOP is set we got problems when locking parts  */
                              /* --------------------------------------*\
                                 For problem determination only ...
                              \*---------------------------------------*/
   If (g.trace?) Then Do
     Call InfMsg("¬ Required Environment variables:"  ,
                 "¬3  TC_USER:" g.TC_USER      ,
                 "¬3  TC_FAMILY:" g.TC_FAMILY    ,
                 "¬3  TC_RELEASE:" g.TC_RELEASE   ,
                 "¬3  TC_WORKAREA:" g.TC_WORKAREA  ,
                 "¬3  TC_CASESENSE:" g.TC_CASESENSE ,
                 "¬3  TC_BUILDPOOL:" g.TC_BUILDPOOL ,
                 "¬3  TWI_RELATIVE:" g.TWI_RELATIVE ,
                 "¬3  TWI_EDITOR:" g.TWI_EDITOR   ,
                 "¬3  TC_TOP: --(cleared)--"  ,
                 "¬ ")
     End
                              /* --------------------------------------*\
                                 Optional information ...
                                  from the TC standard environment
                              \*---------------------------------------*/
   g.TC_BECOME        = Value("TC_BECOME"       ,,"OS2ENVIRONMENT")

   If (g.TC_BECOME <> "") Then g.becomeuser = "-become" g.TC_BECOME
                         Else g.becomeuser = ""
                              /* --------------------------------------*\
                                  ... that is WF integration specific
                              \*---------------------------------------*/
   g.TWI_TARGET_PART  = Value("TWI_TARGET_PART" ,,"OS2ENVIRONMENT")
   g.TWI_RELEASE_PART = Value("TWI_RELEASE_PART",,"OS2ENVIRONMENT")
```
Figure 35 (Part 4 of 25). TWI_DO.CMD

```
                             /* --------------------------------------*\
                                For problem determination only ...
                             \*---------------------------------------*/
 If (g.trace?) Then Do
   Call InfMsg("¬ Optional Environment variables:"       ,
               "¬3  TC_BECOME:" g.TC_BECOME       ,
               "¬3  TWI_TARGET_PART:" g.TWI_TARGET_PART ,
               "¬3  TWI_RELEASE_PART:" g.TWI_RELEASE_PART,
               "¬ ")
   End
                             /* --------------------------------------*\
                                Prepare the -relative parameter
                             \*---------------------------------------*/
 u_tc_relative = Translate(g.twi_relative)
 tc_rel_length = Length(g.twi_relative)
 tc_rel_parm = "-relative" g.twi_relative

/* ------------------------------------------------------------------*\
   Start processing now...
\*-------------------------------------------------------------------*/
 Parse Var allargs  function filelist "(" funcparms
 g.U_function = Translate(function)
                             /* --------------------------------------*\
                                If any of the REQUIRED environment
                                   variables are not set: Stop !
                             \*---------------------------------------*/
 If (XRc <> 0) Then Do
   Call Abort(300+XRc ,
           "Some required environment variables are not set, cannot continue!")
   End
```

Figure 35 (Part 5 of 25). TWI_DO.CMD

```
                            /* ---------------------------------------*\
                               Strip off the projects base path from
                               the filespecs to get the PARTLIST.
                               (Names folded to UPPERCASE for compare)
                            \*---------------------------------------*/
filelist1 = filelist        /* save contents of filelist             */
partlist  = ""
Do While (filelist1 <> "")
  Parse Var filelist1  file filelist1
  If (Left(Translate(file),tc_rel_length) = u_tc_relative)
    Then part = Substr(file,tc_rel_length+1)
    Else part = file
  part = Strip(part,"Leading","\")   /* Note TeamConnection does     */
  part = Strip(part,"Leading","/")   /*  understand / as well as \   */
  partlist = partlist part
  End

/* -----------------------------------------------------------------*\
   Now handle the function requests ...
\*-----------------------------------------------------------------*/
Select
 /* ----------------------------------------------------------------*\
    Edit a set of parts
 \*----------------------------------------------------------------*/
 When (g.U_function = "EDIT") Then Do
  XRc = DoIt("MSG",0,
            ,'START "'g.myname":" g.twi_editor filelist'"',
              "/F /C /MIN" g.mypname "$EDIT" filelist "(" funcparms)
  End /* When */
```

Figure 35 (Part 6 of 25). TWI_DO.CMD

```
/* ---------------------------------------------------------------*\
   Edit a single part:

    - Check out the part
    - Call the editor (it must be a synchronous call!)
    - Check in  the part

   Note: If the $EDIT function is called with a list of files, it will
         invoke the EDIT function using a recursive call !
\*----------------------------------------------------------------*/
When (g.U_function = ″$EDIT″) Then Do
 part = partlist
 file = filelist

 XRc = DoIt(  ″MSG″,0,″TEAMC part -checkout″ part tc_rel_parm)
                           /*--------------------------------------*\
                              Insert (list of) file(s) into editor
                                invocation string replacing ?FILES?
                           \*--------------------------------------*/
 ctl_pos = POS(″?FILES?″,Translate(funcparms))
 If (ctl_pos > 0) Then Do
   funcparms = DelStr(funcparms,ctl_pos,7)
   funcparms = Insert(filelist,funcparms,ctl_pos-1)
   End

″CALL″ g.twi_editor funcparms
                           /* -------------------------------------*\
                              Ask for check in
                           \*--------------------------------------*/
 button = MsgBox(″Do you want to check in″ part,
               ,g.myname″:″ g.U_function,
               ,″YesNo″,″Query″)
 If (Button = ″YES″) Then Do
   XRc = DoIt(  ″MSG″,1,″TEAMC part -checkin″  part tc_rel_parm)
   If (XRc <> 0) Then Do
     Call Abort(″Could not check in″ part ″(″file″) | ″,
               ″Please make sure, that the problem will be solved,″,
               ″and the part is then checked in as soon as possible!″)
     End /* If (XRc... */
   End /* If (MBXrc... */
 End /* When */
```

Figure 35 (Part 7 of 25). TWI_DO.CMD

```
  /* ----------------------------------------------------------------*\
     Define a part
      - We do it unattended if a default component was specified
      - We use the TEAMC GUI call if no default component is given
  \*-----------------------------------------------------------------*/
/*******************************************************************
 **   We do not recommend using this interface, since one copy of     **
 **     the TEAMCGUI program will be loaded for each invocation       **
 **      and will stay in memory even if the TEAMCGUI function        **
 **                     has successfully ended!                       **
 *******************************************************************/
  When (g.U_function = "DEFINE") Then Do
    XRc = DoIt("MSG",0,"START TEAMCGUI -filesfull",
                                   "-relative" g.twi_relative,
                                   "-create",
                                    g.tc_release,
                                    g.tc_workarea,
                                   "-nologo")
   End /* When */
/*******************************************************************/

  /* ----------------------------------------------------------------*\
     CheckOut a part
  \*-----------------------------------------------------------------*/
  When (g.U_function = "CHECKOUT") Then Do
   XRc = DoIt(  "MSG",0,"TEAMC part -checkout"  partlist tc_rel_parm)
   End /* When */

  /* ----------------------------------------------------------------*\
     CheckIn a part
  \*-----------------------------------------------------------------*/
  When (g.U_function = "CHECKIN") Then Do
   XRc = DoIt(  "MSG",0,"TEAMC part -checkin"  partlist tc_rel_parm)
   End /* When */
```

Figure 35 (Part 8 of 25). TWI_DO.CMD

```
/* ---------------------------------------------------------------------*\
   CheckIn a part (with Create and Lock support)

    - try to check in the part
      If the part does not yet exist
         - ask if we should create it
\*---------------------------------------------------------------------*/
When (g.U_function = "CHECKIN+") Then Do

 Do partloop=1 While (partlist <> "")
   Parse Var partlist  part partlist

   msg = "NOMSG"                         /* first time, NO messages */
   Do retryloop=1 To 2 Until (XRc = 0)

     XRc = DoIt(msg,1,"TEAMC part -checkin" part tc_rel_parm)

     msg = "MSG"
                           /* -------------------------------------*\
                              In case of error we get the TC Message
                               code from the first line of the error
                               messages in the TMPFILE
                           \*-------------------------------------*/
     If (XRc <> 0) Then Do
       Parse Value Linein(g.tmpfile,1,1)  With tc_error tc_errtext
       Call Lineout g.tmpfile  /* Close temporary file */

       Select
        /* ---------------------------------------------------------*\
           If Part was not checked out, unlock Part and try again !
        \*---------------------------------------------------------*/
        When (tc_error = tc_err_PartNotCheckedOut  ) Then Do
         Call InfMsg("Part" part "was not checked out,",
                     "trying to lock it first...")
         SRc = DoIt("MSG",1,"TEAMC part -lock" part)
         End /* when */
```

Figure 35 (Part 9 of 25). TWI_DO.CMD

```
        /* -------------------------------------------------------*\
           For any other type of error retrying makes no sense ..
        \*--------------------------------------------------------*/
        Otherwise
         Leave retryloop
         End /* Select */
       End /* If (XRc <> 0) */
     End retryloop
   End partloop
 End /* When */
/* ---------------------------------------------------------------*\
   Show TC Tasklist
\*----------------------------------------------------------------*/
When (g.U_function = ″TCTASK″) Then Do

 XRc = DoIt(″MSG″,0,″START TEAMCGUI -family″ g.tc_family,
                         ″-release″ g.tc_release,
                         ″-workarea″ g.tc_workarea,
                         ″-user″ g.tc_user,
                         g.becomeuser,
                         ″-relative″ g.twi_relative,
                         ″-nologo″)
 End /* When */

/* ---------------------------------------------------------------*\
   Show the parts in the Workarea
\*----------------------------------------------------------------*/
When (g.U_function = ″WASHOW″) Then Do
 XRc = DoIt(″MSG″,0,″START TEAMCGUI -filesfull″,
                         ″-family″ g.tc_family,
                         ″-user″ g.tc_user,
                         g.becomeuser,
                         ″-relative″ g.twi_relative,
                         ″-where ″″releaseName in (′″g.tc_release″′)″,
                         ″and workareaName in (′″g.tc_workarea″′)″,
                         ″order by pathName″″″,
                         ″-nologo″)
 End /* When */
```

Figure 35 (Part 10 of 25). TWI_DO.CMD

```
/* ----------------------------------------------------------------*\
   Analyze Impact of part change
       (for one part only)
\*-----------------------------------------------------------------*/
When (g.U_function = ″IMPACT″) Then Do
 Parse Var partlist  part .
 If (part = ″″) Then Do
   Select
     When (g.TWI_RELEASE_PART <> ″″) Then part = g.TWI_RELEASE_PART
     When (g.TWI_TARGET_PART  <> ″″) Then part = g.TWI_TARGET_PART
     Otherwise
       CALL Abort(2 ″There was no part specified with the function! ¬″,
                    ″Please either select a part, or define a Project″,
                    ″or Release Target.″)
     End /* Select */
   End /* If (part = ″″) */
 XRc = DoIt(″MSG″,1,″TEAMC part -build″  part ″-report″)
 End /* When */

/* ----------------------------------------------------------------*\
   TC Build  (one part only !)

    All BUILD function specifiers look like this: BLDxyyyy...
     where x    is the build type P=part, T=project target, R=release
     and   yyyy may either be FORCed or NORMal


\*-----------------------------------------------------------------*/
When (Left(g.U_function,3) = ″BLD″) Then Do

                           /* ------------------------------------*\
                              Is it a normal or forced build ...
                           \*-------------------------------------*/
 Select
   When (Substr(g.U_function,5,4) = ″FORC″) Then Do
     force = ″-force″
     End
   When (Substr(g.U_function,5,4) = ″NORM″) Then Do
     force = ″″
     End
   Otherwise
     Call Abort(401 ″Unsupported Build function (option):″ g.U_function)
     End /* Select */
```

Figure 35 (Part 11 of 25). TWI_DO.CMD

```
                          /* ---------------------------------------*\
                             Find the right part to be built if
                              the user invoked the PART build...
                          \*----------------------------------------*/
Select
  When (Substr(g.U_function,4,1) = ″P″) Then Do
    Parse Var partlist  part .        /* we can do it for one part */
    part = FindBuildPart(part)
    End /* when */
  When (Substr(g.U_function,4,1) = ″T″) Then Do
    part = g.TWI_TARGET_PART
    End /* when */
  When (Substr(g.U_function,4,1) = ″R″) Then Do
    part = g.TWI_RELEASE_PART
    If (part = ″″) Then part = g.TWI_TARGET_PART
    End /* when */
  Otherwise
    Call Abort(402 ″Unsupported Build function (target):″ g.U_function)
  End /* Select */

If (part = ″″) Then Do
  Call Abort(23 ″Could not determine a part for build function:″,
                g.U_function″. The action is terminated!″)
  End
                          /* ---------------------------------------*\
                             Invoke TC build for the selected target
                          \*----------------------------------------*/
XRc = DoIt(″MSG″,1,″TEAMC part -build″  part , force)
                          /* ---------------------------------------*\
                             Display the build messages
                          \*----------------------------------------*/
Call InfMsg(″¬ Builder messages:″,
            ″¬ ----------------- ¬ ″)
SRc = DoIt(″MSG″,0,″TEAMC part -viewmsg″  part)
                          /* ---------------------------------------*\
                             If build went OK extract objects...
                          \*----------------------------------------*/
If (XRc = 0) Then Do
  Call InfMsg(″ Extracting part″ part ″now ...″)
  SRc = DoIt(″MSG″,0,″TEAMC part -extract″  part tc_rel_parm)
  End
End /* When */
```

Figure 35 (Part 12 of 25). TWI_DO.CMD

```
/* -----------------------------------------------------------------*\
   Touch a part
\*------------------------------------------------------------------*/
When (g.U_function = "TOUCH") Then Do
 XRc = DoIt("MSG",0,"TEAMC part -touch"  partlist)
 End /* When */

/* -----------------------------------------------------------------*\
   View Build Messages
\*------------------------------------------------------------------*/
When (g.U_function = "VIEWBLDMSG") Then Do
 Parse Var partlist  part .        /* we can do it for one part */
 part = FindBuildPart(part)
 XRc = DoIt("MSG",0,"TEAMC part -viewmsg" part)
 End /* When */

/* -----------------------------------------------------------------*\
   Extract part(s)
\*------------------------------------------------------------------*/
When (g.U_function = "EXTRACT") Then Do
 XRc = DoIt("MSG",0,"TEAMC part -extract"  partlist tc_rel_parm)
 End /* When */

/* -----------------------------------------------------------------*\
    Freeze Workarea
\*------------------------------------------------------------------*/
When (g.U_function = "WAFREEZE") Then Do
 XRc = DoIt("MSG",0,"TEAMC workarea -freeze" g.tc_workarea)
 End /* When */

/* -----------------------------------------------------------------*\
   Lock a (list of) part(s)
\*------------------------------------------------------------------*/
When (g.U_function = "LOCK") Then Do
 XRc = DoIt("MSG",0,"TEAMC part -lock"  partlist)
 End /* When */

/* -----------------------------------------------------------------*\
    UnLock a (list of) part(s)
\*------------------------------------------------------------------*/
When (g.U_function = "UNLOCK") Then Do
 XRc = DoIt("MSG",0,"TEAMC part -unlock"  partlist)
 End /* When */
```

Figure 35 (Part 13 of 25). TWI_DO.CMD

```
   /* ---------------------------------------------------------------*\
      Unsupported function
   \*---------------------------------------------------------------*/
   Otherwise
    Say g.myname": Unsupported function" function
    XRc = 99
    End /* Select */

  Call Done(XRc)

 /* -------------------------------------------------------------------*\
    Jump to this point to exit the program
 \*-------------------------------------------------------------------*/
 Done:

  Arg XRc
                             /* -------------------------------------*\
                                If something went on within the EDIT
                                  - keep the session open until user
                                    has seen the error messages.
                                  - If user did nothing for a certain
                                    amount of time show a Pop UP !
                             \*-------------------------------------*/
  If (((g.U_function = "$EDIT"),
      |(g.U_function = "EDIT" )),
     & (XRc <> 0)              ) Then Do
    Call InfMsg(" ¬ ...press any key to end session ...")
    Do loop=1
      If (lines() > 0) Then Leave loop
      Call SysSleep 2
      If (loop = 10)
        Then dy = MsgBox("Something went wrong in the EDIT function,",
                         "please look up the corresponding CMD window",
                         "for more information!",
                        ,g.myname":" g.U_function,
                        ,"ENTER","Exclamation")
      End
    End
```

Figure 35 (Part 14 of 25). TWI_DO.CMD

```
                             /* ---------------------------------------*\
                                This should better be reset in case
                                 the program was interrupted by CTL-C
                             \*---------------------------------------*/
HALT:
 Signal off Error
 Do While (g.exitcmds <> ””)
   Parse Var g.exitcmds  exitcmd ”” g.exitcmds
   Strip(exitcmd)
   End

 Exit XRc

/* ==================================================================*\
 * Start of local subroutines ...
\*==================================================================*/

/* ------------------------------------------------------------------*\
   This routine is going up the build trees in order to find the first
        build part relative to the part specified as a starting point.

   We use the REXX stem PARTLIST to walk up the build LEVELs for a
      maximum of 9.

   If we did not find a build part, a ”?” is returned instead of a part
   name !
\*------------------------------------------------------------------*/
FindBuildPart: Procedure Expose XRc g.
                             /* ---------------------------------------*\
                                Get name of start part and convert the
                                 ’\’ characters to ’/’ for TC Queries!
                             \*---------------------------------------*/
 Parse Arg startpart .
 startpart = Translate(startpart,”/”,”\”)

                             /* ---------------------------------------*\
                                We better clean up the REXX Queue first
                             \*---------------------------------------*/
 Do Queued(); Pull . ; End;
```
Figure 35 (Part 15 of 25). TWI_DO.CMD

```
                            /* --------------------------------------*\
                               Initialize level and partlist
                            \*---------------------------------------*/
buildpart = ""

level = 0
partlist.        = "0"
partlist.level.0 = "1"
partlist.level.1 = startpart

/* ----------------------------------------------------------------*\
   Now start stepping up the build tree level by level...
\*-----------------------------------------------------------------*/
Do level = 0 to 9
  partindex = 0
  nextlevel = level+1
  Do i = 1 to (partlist.level.0)
    mypart = partlist.level.i
    If (g.trace?) Then Do
      Say "Looking at part" mypart "now ..."
      End
                            /* --------------------------------------*\
                               Check if Part is build part ...
                            \*---------------------------------------*/
   "TEAMC Report -raw -view PartView",
                "-where ""nuPathName = '"mypart"'""",
                "-release" g.TC_RELEASE,
                "-workarea" g.TC_WORKAREA "| RXQueue"

    builder1 = ""
    If (Queued())
      Then Parse Pull . "|" . "|" . "|" . "|" . "|",
                      . "|" . "|" . "|" . "|" . "|",
                      . "|" . "|" . "|" . "|" . "|",
                      . "|" . "|" . "|" . "|" builder1 "|" .

    Do Queued()             /* We only expect one line on the queue, */
      Pull . ; End          /*     ... but just be sure !            */
```
Figure 35 (Part 16 of 25). TWI_DO.CMD

```
                        /* --------------------------------------*\
                           If its a build part, we are done !
                        \*---------------------------------------*/
If (builder1 <> "") Then do
  buildpart = mypart
  Leave level /* loop */
  End

/* ----------------------------------------------------------------*\
   It's no build part, so look for any parent parts now..
\*-----------------------------------------------------------------*/
Else Do
  If (g.trace?) Then Do
    Say "   ... was not a builder part !"
    End

 "TEAMC Part -view" mypart ,
            "-long -release" g.TC_RELEASE,
            "-workarea" g.TC_WORKAREA "| RXQueue"
                        /* --------------------------------------*\
                           Save everything in a stem, we need the
                            REXX Queue within the loop again!
                        \*---------------------------------------*/
  QSaved.=""
  QSize  = Queued()
  Do QLine = 1 to QSize
    Parse Pull Qsaved.QLine
    End QLine

  InBuildRelationshipSection = 0

  Do QLine = 1 to Qsize

    If (QSaved.QLine = "build relationships:") Then Do
      InBuildRelationshipSection = 1
      Iterate QLine /* loop */
      End

    If (\InBuildRelationshipSection)
      Then Iterate Qline /* loop */

    If (QSaved.QLine = "versions:")
      Then Leave QLine /* loop */
```
Figure 35 (Part 17 of 25). TWI_DO.CMD

```
/* ---------------------------------------------------------*\
   So let's scan the build relationship lines for build parts
\*----------------------------------------------------------*/
Parse Var QSaved.QLine  partname parttype relation .
                    /*---------------------------------------*\
                       Put potential candidates into partlist
                        for inspection in the next level
                    \*---------------------------------------*/
If (relation = "OutputOf"),
 | (relation = "DependsOn") Then Do
  partindex = partindex + 1
  partname = Strip(partname)
  partlist.nextlevel.partindex = partname
  If (g.trace?) Then Do
    Say "   ... saving for next iteration:" partname
    End
  End
                    /* --------------------------------------*\
                       Have closer look to these immediately !
                         (just for response time reasons)
                    \*---------------------------------------*/
builder2 = ""
If (relation = "OutputOf") Then Do
 "teamc Report -raw -view PartView",
              "-where ""nuPathName = '"partname"'""",
              "-release" g.TC_RELEASE,
              "-workarea" g.TC_WORKAREA "| RXQueue"

  If (Queued())
    Then Parse Pull . "|" . "|" . "|" . "|" . "|",
                    . "|" . "|" . "|" . "|" . "|",
                    . "|" . "|" . "|" . "|" . "|",
                    . "|" . "|" . "|" . "|" builder2 "|" .

  Do Queued()       /* We only expect one line on the queue, */
    Pull . ; End    /*     ... but just be sure !            */
```

Figure 35 (Part 18 of 25). TWI_DO.CMD

```
                                 /* --------------------------------------*\
                                    If its a build part, we are done !
                                 \*---------------------------------------*/
                 If (builder2 <>"") Then Do
                   buildpart = partname
                   Leave level /* loop */
                   End
                 End /* If (relation... */
               End qline /* loop */
             End /* If (builder2 = "") Else */
           End i
           partlist.nextlevel.0 = partindex
         End level /* loop */

       If (buildpart = "")
         Then Say ">>> Could no find a build part for:" startpart
         Else Say ">>> Using build part:" buildpart

       Return buildpart

      /* ------------------------------------------------------------------*\
         Read in TC Environment variable and check if it is set ...
      \*-------------------------------------------------------------------*/
      Get_and_Check_TC_Variable:
       Arg tc_variable .
       g.tc_variable = Value(tc_variable,,"OS2ENVIRONMENT")
       XRc = (g.tc_variable = "")
       If (XRc <> 0) Then Do
         Call ErrMsg(tc_variable "not set!")
         End
       Return XRc
```

Figure 35 (Part 19 of 25). TWI_DO.CMD

```
/* -------------------------------------------------------------------*\
   Execute an OS/2 Program:

     - Saves Error messages in the TMPFILE
     - Checks Rc
     - Display error messages if desired

   Note: We do not pipe the messages to the RXQUEUE service, because
         RXQUEUE would change the return code to 0 !!!
\*--------------------------------------------------------------------*/
DoIt: Procedure Expose g.

 Parse Arg msg , MaxRc , TC_cmd

 If (g.trace?) Then Say TC_Cmd

 Signal off Error
"ERASE" g.tmpfile "2>NUL"    /* Just be sure, erase the file first!   */
 TC_cmd "2>" g.tmpfile       /* Save Error messages into TMPFILE      */
 XRc = Rc
 Signal on  Error
```
Figure 35 (Part 20 of 25). TWI_DO.CMD

```
                              /* ---------------------------------------*\
                                 In case of error, write out the error
                                  messages generated by the command
                              \*---------------------------------------*/
  If (XRc <> 0) Then Do
    If (msg = "MSG") Then Do
      Call ErrMsg("Error ("XRc") executing: ¬ ¬" TC_cmd)
      Say
     "TYPE" g.tmpfile      /* ??? We might check the existence first ! */
      Say
      End
    Else Do
      If (g.trace?) Then Do
        Call InfMsg("Error ("XRc") executing: ¬ ¬" TC_cmd)
        Say
       "TYPE" g.tmpfile        /* ??? We might check the existence first ! */
        Say
        End
      End /* Else Do */
    End /* If (XRc <> 0) Then */

  If (XRc > MaxRc) Then Call Done(XRc)

  Return XRc

 /* ------------------------------------------------------------------*\
    Stop processing issuing an error message and giving back an Rc
 \*------------------------------------------------------------------*/
 Abort: Procedure Expose XRc g.
  Parse Arg retcode message
  Call ErrMsg(message)
  Call Done(retcode)
  Return
```

Figure 35 (Part 21 of 25). TWI_DO.CMD

```
/* -------------------------------------------------------------------*\
 ErrMsg/InfMsg:

   Issue an Error or Informational message
    - Messages are preceeded by the program name
    - Messages are formatted to fit into a 80 Column line
    - Special formatting characters supported: ¬ = Line Break
                                                 = Required Blank
    - ERROR Messages go to SDTERR
\*-------------------------------------------------------------------*/
ErrMsg: Procedure Expose XRc g.
 MsgTarget = "STDERR:"
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

InfMsg: Procedure Expose XRc g.
 MsgTarget = "STDOUT:"
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

WrtMsg: Procedure Expose MsgTarget XRc g.
 Parse Arg msgstring

 nam  = g.myname": "
 nams = Copies(" ",Length(nam))
 max  = 80-(Length(nam))
                             /* -----------------------------------------*\
                                Format the text lines
                             \*-----------------------------------------*/
 n = 1;
 c = 0;
 msgl. = "";
 indent = 0;
 msgl.in.1 = indent;

 Do i = 1 While (msgstring <> "")
   Parse VAR  msgstring  w msgstring
   c = c + Length(w) +1
   Select
```
Figure 35 (Part 22 of 25). TWI_DO.CMD

```
                          /* ---------------------------------------*\
                             Handle Line break and indention
                          \*----------------------------------------*/
      When (Left(w,1) = "¬" /* Indention */) Then Do
        indent = Substr(w,2)
        If (\DataType(indent,"W")) Then indent = 0
        c = 0
        n = n + 1
        msgl.in.n = indent
        End
                          /* ---------------------------------------*\
                             Compose the lines..
                          \*----------------------------------------*/
      Otherwise Do
        If (c <= max) Then Do
          msgl.n = msgl.n w
          End
        Else Do
          n = n + 1
          c = Length(w) +1
          msgl.n = w
          msgl.in.n = indent
          End
        End /* Otherwise */
      End /* Select */
    End i
                          /* ---------------------------------------*\
                             Write the lines ...
                               and handle the required blanks
                          \*----------------------------------------*/
  Call LineOut msgtarget, nam || Copies(" ",msgl.in.1) || Translate(Strip(msgl.1)," ","")
  Do i = 2 to n
    Call LineOut msgtarget, nams || Copies(" ",msgl.in.i) || Translate(Strip(msgl.i)," ","")
    End i

  Return
```

Figure 35 (Part 23 of 25). TWI_DO.CMD

```
/* ===================================================================*\
 * Here we catch all the errors ...
\*====================================================================*/
ERROR:
FAILURE:
NOTREADY: Procedure Expose sigl Rc XRc g.
NOVALUE:
SYNTAX:

 Parse Upper Source myenv mycall mypname .
 Parse Value FileSpec("NAME",mypname) With myname "." .

 traptype = Condition("C")
 calltype = Condition("I")

 Say "REXX trap" calltype "on" traptype "in" mypname ":"

 Say "--->" sigl":" Strip(SourceLine(sigl))
```

Figure 35 (Part 24 of 25). TWI_DO.CMD

```
                                          /* ----------------------------*\
                                             Trap specific processing
                                          \*-----------------------------*/
     Select
      When (traptype = "FAILURE") | (traptype = "ERROR") then do
        Say "Command   :" Condition("D")
        Say "Rc was    :" Rc
        end
      When traptype = "HALT" then do
        Say "Program has been interrupted by an external event!"
        end
      When traptype = "NOVALUE" then do
        Say "Variable  :" Condition("D")
        end
      When traptype = "SYNTAX" then do
        Say "Errortype :" ErrorText(Rc)
        end
      When traptype = "NOTREADY" then do
        Say "Stream    :" Condition("D")
        end
      Otherwise do
        Say "Unknown REXX trap condition >"traptype"<"
        Say "Trap Description :" Condition("D")
        end
      end /* Select */

     Call BEEP 1000,100

    /***
     Say
     Say " Hit any key to continue ..."
     Pull dy
     ***/

     If (calltype = "SIGNAL") Then Exit 998
                              Else Return 999
```

Figure 35 (Part 25 of 25). TWI_DO.CMD

Generating a Project from a TeamConnection Work Area

TWI_PRJ: Creating the WorkFrame Project

The TWI_PRJ REXX program (see Figure 36 on page 129) controls the generation of a WorkFrame project from a TeamConnection work area, using the Project Smarts interface of WorkFrame.

The program basically copies the project template, TWI_PRJ, and adds the values for the variables that describe your TeamConnection work environment.

The installation and user interface of this program are described in "TWI_PRJ Installation Script" on page 41. If you are specifically interested in:

Adding the TeamConnection integration to your Project Smarts catalog
see "Adding Our Project Install Script to Project Smarts" on page 43.

The user interface of the project generation,
see "Creating WorkFrame Project Using Project Smarts" on page 45.

```
/* *******************************************************************
  TeamConnection / WorkFrame Integration

  File ..:  TWI_Prj.cmd
  Project:  Generate WF Project from a TeamConnection Work Area

  Author :  L. Sparmann  (SPARMANN @ BOEVM2)
  Owner .:  L. Sparmann  (SPARMANN @ BOEVM2)

  Description :

   (For further information, please see help!)

  Version and Change History:
  --------------------------*/
  version = "1.00 dated: 02.Apr.96  by: L. Sparmann"
  reason  = "Initial version ..."
/*
********************************************************************/
 Address CMD
"@ECHO OFF"
                                       /* ----------------------------*\
                                         Setup for error trapping
                                       \*----------------------------*/
 Signal on  NOVALUE  ; Signal on  SYNTAX ;           /* SIGNAL only ! */
 Call   on  FAILURE  ; Call   on  ERROR  ;           /* SIGNAL & CALL */
 Call   on  NOTREADY ; Signal on  HALT   ;           /* SIGNAL & CALL */

 /* ------------------------------------------------------------------*\
    Set up the global variables...
 \*------------------------------------------------------------------*/
 XRc = 0

 /* ------------------------------------------------------------------*\
    Basic setup ...
 \*------------------------------------------------------------------*/
 g.            = "?"                   /* All my global variables ... */
                                       /*----------------------------*/
 g.step        = 1                     /* STEP and LASTSTEP are used  */
 g.laststep    = 7                     /*   to indicate the progress. */
                                       /*----------------------------*/
 g.exitcmds    = ""                    /* Cmds to be executed on Exit */
 g.RC_CANCEL   = 95
 g.TWI_EDITOR  = "LXSYNC.CMD"
```

Figure 36 (Part 1 of 22). TWI_PRJ.CMD

```
                            /* ---------------------------------------*\
                               Setup WF specific Variables & Constants
                            \*----------------------------------------*/
stem. = "?"
stem.undefined = stem.undefined
                            /* ---------------------------------------*\
                               Get the call environment
                            \*----------------------------------------*/
Parse Upper Source g.myenv g.mycall g.mypname .
Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
g.mypath  = FileSpec("PATH" ,g.mypname)
g.mydrive = FileSpec("DRIVE",g.mypname)

/* ----------------------------------------------------------------------*\
   We switch on monitoring as early as possible for debug reasons !
\*-----------------------------------------------------------------------*/
XRc = IwfOpenConsole("STEM");
If (XRc <> 0) Then Do
  Call RxMessageBox "IwfOpenConsole failed, Rc was:" XRC,
                   ,g.myname":",
                   ,"ENTER",
                   ,"EXCLAMATION"
  Call Done(XRc)
  End
                            /* ---------------------------------------*\
                               Load OS/2 built-In Utilities
                            \*----------------------------------------*/
If (RxFuncQuery("SysLoadFuncs")) Then Do
  Call RxFuncAdd 'SysLoadFuncs','RexxUtil','SysLoadFuncs'
  Call SysLoadFuncs
  End
                            /* ---------------------------------------*\
                               Load support for Project tools setup
                            \*----------------------------------------*/
If (RxFuncQuery("IwfEnvPrfLoadFuncs")) Then Do
  Call RxFuncAdd "IwfEnvPrfLoadFuncs",'IWFPAPI',"IwfRxEnvPrfLoadFuncs"
  Call IwfEnvPrfLoadFuncs
  Call IwfInitEnvPrfAPIs
  End
                            /* ---------------------------------------*\
                               It's good practice to clean up the
                                REXX Queue in the beginnig
                            \*----------------------------------------*/
Do Queued(); Pull .; End
```

Figure 36 (Part 2 of 22). TWI_PRJ.CMD

```
                            /* --------------------------------------*\
                               Get a system unique name
                                (using the REXX Queue for this)
                            \*---------------------------------------*/
g.myqid = RxQueue("CREATE",)
dyRc    = RxQueue("DELETE",g.myqid)    /* we do not need the Queue ! */

/* -----------------------------------------------------------------*\
   Get Arguments (as typed = mixed case) and provide help if desired
\*------------------------------------------------------------------*/
Parse Arg g.ProjectName, stem.pszCatalog, allopts

If (g.projectname = "?") ,
 | (stem.pszCatalog = "") Then Do
  Call InfMsg("¬ Syntax:" g.myname "project ,catalog <,TRACE>",
              "¬",
              "¬ Additional output for problem determination will be",
                "produced, if the TRACE option is specified with this",
                "command.")
  Call Done(100)
  End

/* -----------------------------------------------------------------*\
   Look for debug or trace setings
\*------------------------------------------------------------------*/
If (POS("TRACE",Translate(allopts)) = 0)
  Then g.trace? = 0
  Else Do
       g.trace? = 1
       Call Value "TWI_TRACE","ON","OS2ENVIRONMENT"
       End
                            /* --------------------------------------*\
                               For problem determination only ...
                            \*---------------------------------------*/
If (g.trace?) Then Do
  Call InfMsg("¬ The following parameters were specified",
              "¬3   Project name:" g.ProjectName ,
              "¬3   Catalog dir.:" stem.pszCatalog  ,
              "¬3   TWI Trace is:" g.trace? ,
              "")
  End
```

Figure 36 (Part 3 of 22). TWI_PRJ.CMD

```
/* ----------------------------------------------------------------*\
   Start processing now...
\*-----------------------------------------------------------------*/
 /* ---------------------------------------------------------------*\
    Get Environment information (to preset variables)
 \*----------------------------------------------------------------*/
 Call UpdateStatus("Querying environment information ...")

 g.TC_USER          = Value("TC_USER"         ,,"OS2ENVIRONMENT")
 g.TC_BECOME        = Value("TC_BECOME"       ,,"OS2ENVIRONMENT")
 g.TC_FAMILY        = Value("TC_FAMILY"       ,,"OS2ENVIRONMENT")
 g.TC_RELEASE       = Value("TC_RELEASE"      ,,"OS2ENVIRONMENT")
 g.TC_WORKAREA      = Value("TC_WORKAREA"     ,,"OS2ENVIRONMENT")
 g.TC_CASESENSE     = Value("TC_CASESENSE"    ,,"OS2ENVIRONMENT")
 g.TC_BUILDPOOL     = Value("TC_BUILDPOOL"    ,,"OS2ENVIRONMENT")
 g.TWI_TARGET_PART  = Value("TWI_TARGET_PART" ,,"OS2ENVIRONMENT")
 g.TWI_TRACE        = Value("TWI_TRACE"       ,,"OS2ENVIRONMENT")
                             /* --------------------------------------*\
                                For problem determination only ...
                             \*---------------------------------------*/
 If (g.trace?) Then Do
   Call InfMsg("¬ Current Environment variable setting:"  ,
               "¬3   TC_USER:"          g.TC_USER       ,
               "¬3   TC_BECOME:"        g.TC_BECOME        ,
               "¬3   TC_FAMILY:"        g.TC_FAMILY     ,
               "¬3   TC_RELEASE:"       g.TC_RELEASE    ,
               "¬3   TC_WORKAREA:"      g.TC_WORKAREA   ,
               "¬3   TC_CASESENSE:"     g.TC_CASESENSE ,
               "¬3   TC_BUILDPOOL:"     g.TC_BUILDPOOL ,
               "¬3   TWI_TARGET_PART:"  g.TWI_TARGET_PART ,
               "")
   End
```

Figure 36 (Part 4 of 22). TWI_PRJ.CMD

```
/* ---------------------------------------------------------------*\
   Query location of base directory for the project
\*----------------------------------------------------------------*/
Call UpdateStatus("Querying Project information...")

stem.pszTargetProject = stem.undefined
stem.pszTargetDirectory = ">>> See Message Log for Info..."

Call InfMsg("¬3 Project (optional):",
            "¬6    You may specify the name of your project here. If",
            "      you leave the field as it is ("stem.pszTargetProject"),",
            "      the release and work",
            "      area name will be used. (You will be asked for it.)",
            "¬3 Directory (required):",
            "¬6    Please specify the drive, that should be used to keep",
            "      the files of your WorkFrame projects.",
            "¬6    A subdirectory structure distinguishing the TC family,",
            "      the workarea and the release will then automatically",
            "      be created.",
            "¬3 Folder (required):",
            "¬6    You may select a folder where your project Icon will",
            "      be placed. (The OS/2 DESKTOP is the default).",
            "")

Do Until (Rc = 0)
   Rc = IwfQueryLocation("STEM");
   If (Rc = g.RC_CANCEL)
     Then Call Cancel
     Else If (Rc <> 0)
            Then Call Abort(Rc "¬ Error querying Target information.",
                               "¬ IwfQueryLocation Rc was:" Rc);
  End

stem.pszTargetDirectory = Strip(stem.pszTargetDirectory,"TRAILING","\")
                            /* ---------------------------------------*\
                               For problem determination only ...
                            \*----------------------------------------*/
If (g.trace?) Then Do
  Call InfMsg("¬ Target Information:"  ,
              "¬3   TargetFolder:" stem.pszTargetFolder,
              "¬3   TargetProject:" stem.pszTargetProject,
              "¬3   TargetDirectory:" stem.pszTargetDirectory,
              "")
  End
```

Figure 36 (Part 5 of 22). TWI_PRJ.CMD

```
/* -----------------------------------------------------------------*\
   Ask for basic TC environment definition...
\*------------------------------------------------------------------*/
Call UpdateStatus("Querying basic TC information...")

Call InfMsg("¬ Please specify/modify your basic Teamconnection environment",
            "  by defining values for the environment variables in the list...",
            "¬ The (R) on the left hand side of the variable description",
            "  indicates that a value is required, an (O) indicates",
            "  optional variables.",
            "")
                            /* ---------------------------------------*\
                               Preset Variables
                            \*----------------------------------------*/
If (g.TC_CASESENSE = "") Then g.TC_CASESENSE = "LOWER"

stem.usVariableCount = 0
stem.pszApplication  = g.myname"1"

Call DefVar "TC_FAMILY       ",,
            "(R) The Name of the TeamConnection Family."        ,,
             g.TC_FAMILY
Call DefVar "TC_USER         ",,
            "(R) Your TeamConnection UserId."                   ,,
             g.TC_USER
Call DefVar "TC_BECOME       ",,
            "(O) The BECOME UserId (if necessary)."             ,,
             g.TC_BECOME
Call DefVar "TC_CASESENSE    ",,
            "(R) How should TC handle Uppercase names ?"        ,,
             g.TC_CASESENSE
Call DefVar "TWI_TRACE       ",,
            "(O) Trace switch for TWI_... programs (ON/OFF)."   ,,
             g.TWI_TRACE
                            /*----------------------------------------*\
                               Get Environment information from user
                            \*----------------------------------------*/
Call GetEnvironmentInfo
```

Figure 36 (Part 6 of 22). TWI_PRJ.CMD

```
/* ------------------------------------------------------------------*\
   Ask for project specific information...
\*-------------------------------------------------------------------*/
Call UpdateStatus(″Querying project specific information...″)

Call InfMsg(″¬ Please define the project specific information by setting″,
            ″  values for the environment variables in the list...″,
            ″¬ The (R) on the left hand side of the variable description″,
            ″  indicates that a value is required, an (O) indicates″,
            ″  optional variables.″,
            ″″)
                            /* --------------------------------------*\
                               Preset Variables
                            \*---------------------------------------*/
stem.usVariableCount = 0
stem.pszApplication  = g.myname″2″

Call DefVar ″TWI_WORK_PARTS  ″,,
            ″(O) A list of parts you plan to work on.″          ,,
             g.TWI_WORK_PARTS
Call DefVar ″TWI_TARGET_PART ″,,
            ″(O) The Build Target for this Work Area″           ,,
             g.TWI_TARGET_PART
Call DefVar ″TC_WORKAREA     ″,,
            ″(R) The Name of the Work Area you plan to work on.″,,
             g.TC_WORKAREA
Call DefVar ″TC_RELEASE      ″,,
            ″(R) The Name of the Release you are working with.″ ,,
             g.TC_RELEASE
Call DefVar ″TC_BUILDPOOL    ″,,
            ″(R) The Name of the TC Build Pool you want to use.″,,
             g.TC_BUILDPOOL
                            /*---------------------------------------*\
                               Get Environment information from user
                            \*---------------------------------------*/
Call GetEnvironmentInfo
```

Figure 36 (Part 7 of 22). TWI_PRJ.CMD

```
/* ------------------------------------------------------------------*\
   Set TWI_RELATIVE according to our naming convention
     (family-workarea-release) and take care of '.' and '\' in names
\*-------------------------------------------------------------------*/
g.TWI_RELATIVE = stem.pszTargetDirectory"\",
               || Translate(g.TC_FAMILY    ,"--",".\")"\",
               || Translate(g.TC_WORKAREA  ,"--",".\")"\",
               || Translate(g.TC_RELEASE   ,"--",".\")

Call Value "TWI_RELATIVE",g.TWI_RELATIVE,"OS2ENVIRONMENT"
                            /* --------------------------------------*\
                               For problem determination only ...
                            \*---------------------------------------*/
If (g.trace?) Then Do
  Call InfMsg("¬ TWI_RELATIVE has been set to: ¬3 " g.TWI_RELATIVE )
  End

/* ------------------------------------------------------------------*\
   Check if the project target directory is still empty
\*-------------------------------------------------------------------*/
Call SysFileTree g.twi_relative"\*","filelist","FSO"

If (filelist.0 <> 0) Then Do
  Call InfMsg("¬ >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>",
              "¬ There are already files existing in the",
              "  directory tree below the Project Base Directory",
              "  ("g.twi_relative").",
              "¬ In this case, an automatic extraction of files from",
              "  the TeamConnection database might/%var be dangerous",
              "  and is therefore not supported!",
              "¬ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
  End
Else Do
```

Figure 36 (Part 8 of 22). TWI_PRJ.CMD

```
   /* ----------------------------------------------------------------*\
      If there are no files yet, we offer a build tree extraction
   \*-----------------------------------------------------------------*/
   If (g.TWI_WORK_PARTS = "") ,
    | (g.TWI_WORK_PARTS = g.undefined) Then Do
                             /*---------------------------------------*\
                                If no work parts specified, start GUI
                             \*---------------------------------------*/
     Call InfMsg("Asking user to extract his Build Target(s)...")

     Call InfMsg("¬ The TeamConnection GUI is started now, to allow you to",
                 "  extract the files you plan to work on, from the TC",
                 "¬ Work Area. If you are done with this, please close the",
                 "  TC GUI, so that the WF Project generation can continue.",
                 "¬ We will later offer you an automatic extract of all the",
                 "  files belongnig to the build trees that is defined by",
                 "  the files you initially extracted.",
                 "")

     If (g.TC_BECOME <> "") Then becomeuser = "-become" g.TC_BECOME
                            Else becomeuser = ""

    "TEAMCGUI -family" g.tc_family,
             "-release" g.tc_release,
             "-workarea" g.tc_workarea,
             "-user" g.tc_user,
               becomeuser,
             "-relative" g.twi_relative,
             "-nologo"
     End /* If (g.TWI_WORK_PARTS ... */
```
Figure 36 (Part 9 of 22). TWI_PRJ.CMD

```
  /* ---------------------------------------------------------------*\
     Call build tree extraction service (TW_XBT)

       The environment is already prepared:
         - TC_... variables are set
         - TWI_RELATIVE is set to project base directory
         - TWI_WORK_PARTS might be set by the user

  \*---------------------------------------------------------------*/
 ”@ECHO ON”
  Say
 'START ”Extract parts for Work Area:' g.TC_WORKAREA'”',
       '/C /MIN TWI_XBT.CMD'
  Say
 ”@ECHO OFF”
  End /* If (filelist.0 <> 0) Else */

/* -----------------------------------------------------------------*\
   If the user did not define a project title, we now set the
    title to: WorkAreaName (ReleaseName)
\*-----------------------------------------------------------------*/
If (stem.pszTargetProject = stem.undefined)
  Then stem.pszTargetProject = g.TC_WORKAREA||Newline()||”(”g.TC_RELEASE”)”

/* -----------------------------------------------------------------*\
   Now create new WF project by copying the template
     (we assume we hooked in to the VACPP projects ...)
\*-----------------------------------------------------------------*/
Call UpdateStatus(”The project will be created in Folder:”,
                   stem.pszTargetFolder)
                            /* -------------------------------------*\
                               Find location of template project
                            \*-------------------------------------*/
TemplatePath = g.mydrive||Strip(g.mypath,”Trailing”,”\”)
If (TemplatePath = ””)
  Then Call Abort(99 ”Error obtaining Project Smarts template path.”);
```

Figure 36 (Part 10 of 22). TWI_PRJ.CMD

```
                          /* -------------------------------------*\
                             Copy the template project now
                          \*--------------------------------------*/
stem.pszSourceProject     = TemplatePath"\"g.myname
/*   pszTargetProject     = has been defined earlier in this program */
/*   pszTargetDirectory   = has been defined earlier in this program */
/*   pszTargetFolder      = has been defined earlier in this program */
stem.pszTargetProjectSetup= "TARGETNAME=",
                          ||   Translate(g.TWI_TARGET_PART,"\","/")";",
                          || "PAMLOCATION:IWFBPAM="g.TWI_RELATIVE";",
                          || "PAMDEFAULT:IWFBPAM="g.TWI_RELATIVE";",
                          || "INHERITLIST=<CPPDFTPRJ>;",
                          || "TITLE="stem.pszTargetProject";",
                          || ""

If (g.trace?) Then Do
  Call InfMsg("¬ Object Setupstring:",
            "¬3" stem.pszTargetProjectSetup)
  End

XRc = IwfCreateProjectFromProject("STEM")
If (XRc <> 0)
  Then Call Abort(XRc "IwfCreateProjectFromProject failed with Rc=" XRc);

/* -----------------------------------------------------------------*\
   Create Variables ...
\*------------------------------------------------------------------*/
stem.pszProject = stem.pszTargetFile  /* Undocumented return value of*/
                                      /*  IwfCreateProjectFromProject*/

Call AddVar "TC_USER         " , g.TC_USER
Call AddVar "TC_BECOME       " , g.TC_BECOME
Call AddVar "TC_FAMILY       " , g.TC_FAMILY
Call AddVar "TC_RELEASE      " , g.TC_RELEASE
Call AddVar "TC_WORKAREA     " , g.TC_WORKAREA
Call AddVar "TC_CASESENSE    " , g.TC_CASESENSE
Call AddVar "TC_BUILDPOOL    " , g.TC_BUILDPOOL
Call AddVar "TWI_EDITOR      " , g.TWI_EDITOR
Call AddVar "TWI_RELATIVE    " , g.TWI_RELATIVE
Call AddVar "TWI_TARGET_PART " , g.TWI_TARGET_PART
Call AddVar "TWI_TRACE       " , g.TWI_TRACE

Call Done(0)
```

Figure 36 (Part 11 of 22). TWI_PRJ.CMD

```
/* ------------------------------------------------------------------*\
   Jump to this point to exit the program
\*-------------------------------------------------------------------*/
Done:
 Call UpdateStatus("The project template generation has completed")

 Arg XRc
 Call RxMessageBox "The Workframe project installation has finished.",
                   "Return Code was" XRc "!",
                   ,g.myname":",
                   ,"OK",
                   ,"INFORMATION"
                             /* -------------------------------------*\
                                This should better be reset in case
                                 the program was interrupted by CTL-C
                             \*--------------------------------------*/
HALT:
                             /* -------------------------------------*\
                                Close Installation Console ...
                             \*--------------------------------------*/
 dyRc = IwfTermEnvPrfAPIs()
 dyRc = IwfEnvPrfDropFuncs()
 dyRc = IwfCloseConsole("STEM");

 Exit XRc

/* ==================================================================*\
 * Start of local subroutines ...
\*===================================================================*/
```

Figure 36 (Part 12 of 22). TWI_PRJ.CMD

```
/* -------------------------------------------------------------------*\
   Define Workframe Variable
\*--------------------------------------------------------------------*/
DefVar: Procedure Expose XRc g. stem.

 Parse Arg VarName , VarDescr , VarValue

 i = stem.usVariableCount + 1

 stem.pszVariableName.i         = "%"Strip(VarName)"%"
 stem.pszVariableDescription.i  = Strip(VarDescr)
 stem.pszVariableValue.i        = Strip(VarValue)

 stem.usVariableCount = i

 Return

/* -------------------------------------------------------------------*\
   This one just generated a NewLine charater used in the setup strings
\*--------------------------------------------------------------------*/
NewLine:
 Return '0a'x

CRLF:
 Return '0d0a'x
```

Figure 36 (Part 13 of 22). TWI_PRJ.CMD

```
/* -------------------------------------------------------------------*\
   Update the status line and write a log entry indicating the current
   step of the project setup...
\*--------------------------------------------------------------------*/
UpdateStatus: Procedure Expose XRc g. stem.

 Parse Arg text

 g.step = g.step + 1
 If (g.step > g.laststep) Then Do
   Call ErrMsg("too many steps, please update G.LASTSTEP to:" g.step "!")
   stem.usPercent = g.laststep
   End
 Else stem.usPercent = (100 / g.laststep * g.step) % 1

 rc = IwfUpdateConsoleProgress("STEM");
 If (Rc <> 0) Then Do
   Call ErrMsg("IwfUpdateConsoleStatus failed, Rc was:" Rc)
   End

 Call InfMsg("("g.step")" text)
 stem.pszStatusText = text
 rc = IwfUpdateConsoleStatus("STEM");
 If (Rc <> 0) Then Do
   Call ErrMsg("IwfUpdateConsoleStatus failed, Rc was:" Rc)
   End
 Return
```

Figure 36 (Part 14 of 22). TWI_PRJ.CMD

```
/* ------------------------------------------------------------------*\
   Add Environment variable
\*-------------------------------------------------------------------*/
AddVar:  Procedure Expose XRc g. stem.

 Parse Arg VarName , VarValue

 stem.pszName   = Strip(VarName)

 If (Strip(VarValue) = "")
   Then stem.pszValue  = " "     /* WF does not handle ''  correctly! */
   Else stem.pszValue  = Strip(VarValue)

 Rc = IwfAddVariable("STEM")

 If (Rc <> 0)
   Then Call Abort Rc "Could not Add Variable:" stem.pszName "(Rc="Rc")"

 Return

/* ------------------------------------------------------------------*\
   Get environment information from user ...

     - Restore variable set from catalog
     - Start Variable Dialog
     - Check that all required variables are set (R)
     - Set global variables (g.) from user input
     - Save variable set in catalog
     - Set OS/2 Environment Variables from user input

\*-------------------------------------------------------------------*/
GetEnvironmentInfo: Procedure Expose XRc g. stem.
                             /* -------------------------------------*\
                                Try to restore variables from catalog
                             \*--------------------------------------*/
 If (g.trace?) Then Do
   Call InfMsg("¬ Trying to retieve Environment Variable set",
                stem.pszApplication,
             "¬ from Catalog" stem.pszCatalog)
   End
```

Figure 36 (Part 15 of 22). TWI_PRJ.CMD

```
                            /* --------------------------------------*\
                               This fixes some IwfXxxxVariables bugs:
                                - empty variables get %variablename%
                                  as their value
                                - backsalash (\) is trated as C control
                                  character by IwfQueryVariables. Use /
                            \*---------------------------------------*/
     Rc = IwfRestoreVariables("STEM");
     If (Rc <> 0) Then Do
       Call InfMsg("¬  Error restoring variable settings.",
                   "¬3 IwfRestoreVariables Rc was:" Rc".",
                   "¬3 ...default values will be used.")
       End

     Do i=1 to stem.usVariableCount
       If (stem.pszVariableName.i = stem.pszVariableValue.i)
         Then stem.pszVariableValue.i =""
       End i
                            /* --------------------------------------*\
                               Open variable dialog
                            \*---------------------------------------*/
     Do loop = 1 Until (AllRequiredVarsSet?)
       Do Until (Rc = 0)
         Rc = IwfQueryVariables("STEM");
         If (Rc = g.RC_CANCEL)
           Then Call Cancel
           Else If (Rc <> 0)
                  Then Call Abort(Rc "¬ Error querying variable settings.",
                              "¬ Return Code of IwfQueryVariables was:" Rc );
       End
                            /* --------------------------------------*\
                               For problem determination only ...
                            \*---------------------------------------*/
       If (g.trace?) Then Do
         Call InfMsg("¬ Current Environment Setting:"  ,
                     "")
         Do i=1 to stem.usVariableCount
           Say "  " stem.pszVariableName.i":" stem.pszVariableValue.i
           Say "        ("stem.pszVariableDescription.i")"
           End i
         End
```
Figure 36 (Part 16 of 22). TWI_PRJ.CMD

```
                              /* --------------------------------------*\
                                 Check that all required variables
                                  have been set by the user !
                              \*---------------------------------------*/
     AllRequiredVarsSet? = 1
     Do i=1 to stem.usVariableCount
       If (Left(stem.pszVariableDescription.i,3) = ″(R)″),
        & (stem.pszVariableValue.i = stem.pszVariableDescription.i) Then Do
         AllRequiredVarsSet? = 0
         Call InfMsg(″¬ There is at least one of the required (R) variables″,
                     ″  not set!″,
                     ″″)
         Call RxMessageBox ″There is at least one of the required (R)″,
                           ″variables not set!″,
                           ,g.myname″:″,
                           ,″ENTER″,
                           ,″EXCLAMATION″
         Leave i
         End
       End i
     End loop
                              /* --------------------------------------*\
                                 Set the G.xxx and the OS/2 environment
                                  variables from the users input
                                   - take care of empty variables)
                              \*---------------------------------------*/
   trc_lines = ″″
   Do i=1 to stem.usVariableCount
     VarName = Strip(stem.pszVariableName.i,″BOTH″,″%″)
     If (stem.pszVariableValue.i <> stem.pszVariableDescription.i)
       Then g.VarName = Strip(Translate(stem.pszVariableValue.i,″  ″,CRLF()))
       Else g.VarName = ″″
     Call Value varname , g.varname , ″OS2ENVIRONMENT″
     stem.pszVariableValue.i = g.varname
     stem.pszVariableValue.i = Translate(g.varname,″/″,″\″)
     trc_lines = trc_lines ″¬3 g.″varname″:″ g.varname
     End i

   If (g.trace?) Then Do
     Call InfMsg(″¬ Final Environment definition:″ trc_lines)
     End
```

Figure 36 (Part 17 of 22). TWI_PRJ.CMD

```
                              /* --------------------------------------*\
                                 Save variable settings in catalog
                              \*---------------------------------------*/
  If (g.trace?) Then Do
    Call InfMsg("¬ Trying to save Environment Variable set",
                   stem.pszApplication,
                "¬ into Catalog" stem.pszCatalog)
    End

  Rc = IwfSaveVariables("STEM");
  If (Rc <> 0)
    Then Call InfMsg("...  Error saving variable settings.",
                     "¬3 IwfSaveVariables Rc was:" Rc".")

  Return

 /* ----------------------------------------------------------------------*\
    Confirm Cancellation
 \*-----------------------------------------------------------------------*/
 Cancel:
  rc = RxMessageBox("Do you really want to cancel?",
                    ,g.myname":",
                    ,"YESNO",
                    ,"QUERY")

  If (rc = 6 /*YES*/) Then Do
    Call InfMsg("....cancelled by the user !")
    Call Done(8)
    End

  Return;

 /* ----------------------------------------------------------------------*\
    Stop processing issuing an error message and giving back an Rc
 \*-----------------------------------------------------------------------*/
 Abort: Procedure Expose XRc g. stem.
  Parse Arg retcode message
  Call ErrMsg(message)
  Call Done(retcode)
  Return
```

Figure 36 (Part 18 of 22). TWI_PRJ.CMD

```
/* -----------------------------------------------------------------*\
 ErrMsg/InfMsg:

   Issue an Error or Informational message
    - Messages are preceeded by the program name
    - Messages are formatted to fit into a 80 Column line
    - Special formatting characters supported:
         ¬n = Line Break and set line indentation level to n (0=default)
           = Required Blank
    - ERROR Messages go to SDTERR
\*------------------------------------------------------------------*/
ErrMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

InfMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

WrtMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring

 nam  = g.myname″: ″
 nams = Copies(″ ″,Length(nam))
 max  = 75-(Length(nam))
                             /* -------------------------------------*\
                                Format the text lines
                             \*--------------------------------------*/
 n = 1;
 c = 0;
 msgl. = ″″;
 indent = 0;
 msgl.in.1 = indent;

 Do i = 1 While (msgstring <> ″″)
   Parse VAR  msgstring  w msgstring
   c = c + Length(w) +1
   Select
```
Figure 36 (Part 19 of 22). TWI_PRJ.CMD

```
                              /* ---------------------------------------*\
                                 Handle Line break and indention
                              \*----------------------------------------*/
    When (Left(w,1) = "¬" /* Indention */) Then Do
      indent = Substr(w,2)
      If (\DataType(indent,"W")) Then indent = 0
      c = 0
      n = n + 1
      msgl.in.n = indent
      End
                              /* ---------------------------------------*\
                                 Compose the lines..
                              \*----------------------------------------*/
    Otherwise Do
      If (c <= max) Then Do
        msgl.n = msgl.n w
        End
      Else Do
        n = n + 1
        c = Length(w) +1
        msgl.n = w
        msgl.in.n = indent
        End
      End /* Otherwise */
    End /* Select */
  End i
                              /* ---------------------------------------*\
                                 Write the lines ...
                                   and handle the required blanks
                              \*----------------------------------------*/
Say
Say nam || Copies(" ",msgl.in.1) || Translate(Strip(msgl.1)," ","")
Do i = 2 to n
  Say nams || Copies(" ",msgl.in.i) || Translate(Strip(msgl.i)," ","")
  End i

Return
```

Figure 36 (Part 20 of 22). TWI_PRJ.CMD

```
/* ===================================================================*\
 * Here we catch all the errors ...
\*====================================================================*/
ERROR:
FAILURE:
NOTREADY: Procedure Expose sigl Rc XRc g. stem.
NOVALUE:
SYNTAX:

 Parse Upper Source myenv mycall mypname .
 Parse Value FileSpec("NAME",mypname) With myname "." .

 traptype = Condition("C")
 calltype = Condition("I")

 Say "REXX trap" calltype "on" traptype "in" mypname ":"

 Say "--->" sigl":" Strip(SourceLine(sigl))
```

Figure 36 (Part 21 of 22). TWI_PRJ.CMD

```
                                            /* ----------------------------*\
                                               Trap specific processing
                                            \*-----------------------------*/
     Select
      When (traptype = ″FAILURE″) | (traptype = ″ERROR″) then do
        Say ″Command   :″ Condition(″D″)
        Say ″Rc was    :″ Rc
        end
      When traptype = ″HALT″ then do
        Say ″Program has been interrupted by an external event!″
        end
      When traptype = ″NOVALUE″ then do
        Say ″Variable  :″ Condition(″D″)
        end
      When traptype = ″SYNTAX″ then do
        Say ″Errortype :″ ErrorText(Rc)
        end
      When traptype = ″NOTREADY″ then do
        Say ″Stream    :″ Condition(″D″)
        end
      Otherwise do
        Say ″Unknown REXX trap condition >″traptype″<″
        Say ″Trap Description :″ Condition(″D″)
        end
      end /* Select */

     Call BEEP 1000,100

     If (calltype = ″SIGNAL″) Then Exit 998
                              Else Return 999
```

Figure 36 (Part 22 of 22). TWI_PRJ.CMD

TWI_XBT: Extracting a Whole Build Tree

The TWI_XBT program (see Figure 37 on page 152) takes a list of parts and for every part climbs the TeamConnection build tree to find the “nearest” build part. For each build part identified in this way, the program extracts all of the parts of its build tree and places them in a directory structure according to the value of the OS/2 environment variable, `TWI_RELATIVE`.

The list of parts to start with is either the list of files in the OS/2 environment variable, `TWI_WORK_PARTS`, or the files that already exist in the directory tree below the path specified in `TWI_RELATIVE`.

TWI_XBT basically uses the TeamConnection command line interface (-report), the REXX queue interface (RxQueue), and REXX STEM variables to perform its task.

```
/* *******************************************************************
  TeamConnection / WorkFrame Integration

  File ..:  TWI_XBT.CMD
  Project:  Extract a build tree based on a part

  Author :  L. Sparmann  (SPARMANN @ BOEVM2)
  Owner .:  L. Sparmann  (SPARMANN @ BOEVM2)

  Description :

   Extract TC Build Tree(s) belonging to a set of files specified in
   the OS/2 Environment Variable TWI_WORK_PARTS. If this variable is
   not set, and there are files found in the directory structure below
   the base directory specified in the OS/2 Environment Variable
   TWI_RELATIVE, the build trees are extracted based on this set of
   files.

   (For further information, please have a look to the code ...)

  Version and Change History:
  --------------------------*/
  version = "1.00 dated:  2.Apr.96  by: L. Sparmann"
  reason  = "Initial version ..."
/*

********************************************************************/
 Address CMD
"@ECHO OFF"
                                       /*-----------------------------*\
                                         Setup for error trapping
                                       \*-----------------------------*/

 Signal on  NOVALUE  ; Signal on  SYNTAX ;           /* SIGNAL only ! */
 Call   on  FAILURE  ; Call   on  ERROR  ;           /* SIGNAL & CALL */
 Call   on  NOTREADY ; Signal on  HALT   ;           /* SIGNAL & CALL */

 /* ---------------------------------------------------------------*\
    Set up the global variables...
 \*----------------------------------------------------------------*/
 XRc = 0
```

Figure 37 (Part 1 of 14). TWI_XBT.CMD

```
/* -----------------------------------------------------------------*\
   Basic setup ...
\*------------------------------------------------------------------*/
g.            = "?"                  /* All my global variables ... */
                            /* --------------------------------------*\
                               Get the call environment
                            \*---------------------------------------*/
Parse Upper Source g.myenv g.mycall g.mypname .
Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
g.mypath  = FileSpec("PATH" ,g.mypname)
g.mydrive = FileSpec("DRIVE",g.mypname)

/* -----------------------------------------------------------------*\
   Setup the environment ...
\*------------------------------------------------------------------*/
                            /* --------------------------------------*\
                               Load OS/2 built-In Utilities
                            \*---------------------------------------*/
If (RxFuncQuery("SysLoadFuncs")) Then Do
  Call RxFuncAdd 'SysLoadFuncs','RexxUtil','SysLoadFuncs'
  Call SysLoadFuncs
  End
                            /* --------------------------------------*\
                               It's good practice to clean up the
                                REXX Queue in the beginnig
                            \*---------------------------------------*/
Do Queued(); Pull .; End
                            /* --------------------------------------*\
                               Look for debug or trace setings
                            \*---------------------------------------*/
If (Value("TWI_TRACE",,"OS2ENVIRONMENT") = "ON") Then g.trace? = 1
                                                 Else g.trace? = 0
```

Figure 37 (Part 2 of 14). TWI_XBT.CMD

```
  /* -------------------------------------------------------------------*\
     Get Arguments (as typed = mixed case) and provide help if desired
  \*--------------------------------------------------------------------*/
  Parse Arg allargs

  If (allargs <> ””) Then Do
    Call InfMsg(”¬ Syntax:” g.myname
                ”¬”,
                ”¬ For function description see file header ...”,
                ”¬”,
                ”¬ Additional output for problem determination will be”,
                ”  produced, if the TWI_TRACE environment variable”,
                ”  is set to ON.”)
    Call Done(100)
    End

 /* ---------------------------------------------------------------------*\
    Start processing now...
 \*----------------------------------------------------------------------*/
                             /*---------------------------------------*\
                                Get Arguments from OS/2 Environment
                             \*---------------------------------------*/
  g.TWI_RELATIVE     = Value(”TWI_RELATIVE”    ,,”OS2ENVIRONMENT”)
  g.TWI_WORK_PARTS   = Value(”TWI_WORK_PARTS”  ,,”OS2ENVIRONMENT”)

  g.TC_RELEASE       = Value(”TC_RELEASE”      ,,”OS2ENVIRONMENT”)
  g.TC_WORKAREA      = Value(”TC_WORKAREA”     ,,”OS2ENVIRONMENT”)

  /* -------------------------------------------------------------------*\
     If Work Parts specified, generate a list (stem)
  \*--------------------------------------------------------------------*/
  If (g.TWI_WORK_PARTS <> ””) Then Do
    i = 0
    Do While (g.TWI_WORK_PARTS <> ””)
      Parse Var g.TWI_WORK_PARTS part g.TWI_WORK_PARTS
      i = i + 1
      workpart.i = Strip(part)
      End /* Do While */
    workpart.0 = i
    End
```

Figure 37 (Part 3 of 14). TWI_XBT.CMD

```
/* ----------------------------------------------------------------*\
   If there were not Work Parts specifies, inspect the TWI_Relative
    directory for files to be taken as a work parts
\*-----------------------------------------------------------------*/
Else Do
  Call SysFileTree g.twi_relative"\*","workpart","FSO"
  End

If (workpart.0 = 0) Then Do
  button = MsgBox("There was not list of workparts specified !",
                  ,g.myname":",
                  ,"OK",
                  ,"EXCLAMATION")
  Call Done(99)
  End

/* ----------------------------------------------------------------*\
   Convert file names into TC part names
\*-----------------------------------------------------------------*/
u_base = Translate(g.twi_relative)
Do i = 1 to workpart.0
  file   = workpart.i
  u_file = Translate(file)
  If (Left(u_file,Length(u_base)) = u_base)
    Then file = Substr(file,Length(u_base)+1)
  file = Translate(file,"/","\")    /* no '\' allowed in TC queries! */
  workpart.i = Strip(file,"Leading","/")
  End i
                            /*---------------------------------------*\
                               For problem determination only
                            \*---------------------------------------*/
If (g.trace?) Then Do
  Call InfMsg("¬ This is the list of Work Parts:")
  Do i = 1 to workpart.0
    Say "         " workpart.i
    End i
  End

/* ----------------------------------------------------------------*\
   Ask the user to extract the build target from the TC Workarea ...
    (this will allow us to extract the files from TC to WF)
\*-----------------------------------------------------------------*/
```

Figure 37 (Part 4 of 14). TWI_XBT.CMD

```
                                 /* --------------------------------------*\
                                    Ask if user wants to do the extract
                                 \*---------------------------------------*/
     button = MsgBox("Do you want me to Extract the build tree(s) ?",
                     ,g.myname":",
                     ,"YESNO",
                     ,"QUERY")

     /* ---------------------------------------------------------------------*\
        Find the build parts and extract the build trees
     \*----------------------------------------------------------------------*/
     If (button = "YES") Then Do
       p = 0                  /* this counts packages */
       fmax = 11              /* Maximum package size */
       f = fmax               /* this counts files in package */
       XPack. = ""            /* Extract Packages */
       Do wpi = 1 to workpart.0
         partname = workpart.wpi
         If (g.btree.partname.InList? = 1)  Then Iterate wpi

         buildpart = FindBuildPart(partname)

         If (g.btree.buildpart.InList? = 1) Then Iterate wpi
         If (buildpart = "")            Then Iterate wpi
                                 /* --------------------------------------*\
                                    Find parts of Build Tree
                                 \*---------------------------------------*/
        "TEAMC Report -raw -view bPartView",
                          "-where ""nuPathName='"buildpart"'""",
                          "| RxQueue"
```
Figure 37 (Part 5 of 14). TWI_XBT.CMD

```
                                  /* --------------------------------------*\
                                     Sort out duplicates and create packages
                                  \*---------------------------------------*/
         If (Queued() > 0) Then Do
           Do Queued()
             Parse Pull . ″|″ . ″|″ . ″|″ . ″|″ . ″|″,
                        . ″|″ . ″|″ . ″|″ . ″|″ . ″|″,
                        . ″|″ . ″|″ partname ″|″ .
             If (g.btree.partname.InList? = g.undefined) Then Do
               f = f + 1
               If (f > fmax) Then do
                 f = 1
                 p = p + 1
                 End
               XPack.p = XPack.p partname
               g.btree.partname.InList? = 1
               End
             End /* Do Queued() */
           End /* If (Queued() > 0) */
         End wpi
                                  /* --------------------------------------*\
                                     Extract the parts
                                  \*---------------------------------------*/
       Call InfMsg(″¬ Starting extract of parts... ¬″)
      ″@ECHO ON″

       Do i=1 to p
        ″TEAMC Part -extract″ XPack.i ,
                   ″-relative″ g.twi_relative
         XRc = XRc + Rc
         End i
       End /* If (button = ″YES″) */

     Call Done(0)
```

Figure 37 (Part 6 of 14). TWI_XBT.CMD

```
/* -------------------------------------------------------------------*\
   Jump to this point to exit the program
\*--------------------------------------------------------------------*/
Done:

 Arg XRc
 Call MsgBox "Extracting the files from TC Work Area" g.TC_WORKAREA,
             "Ended with a return code of" XRc ".",
             ,g.myname":",
             ,"OK",
             ,"INFORMATION"
                             /* -------------------------------------*\
                                This should better be reset in case
                                 the program was interrupted by CTL-C
                             \*--------------------------------------*/
HALT:

 Exit XRc

/* ===================================================================*\
 * Start of local subroutines ...
\*====================================================================*/
/* -------------------------------------------------------------------*\
   This routine is going up the build trees in order to find the first
        build part relative to the part specified as a starting point.

   We use the REXX stem PARTLIST to walk up the build LEVELs for a
      maximum of 9.

   If we did not find a build part, a "?" is returned instead of a part
   name !
\*--------------------------------------------------------------------*/
FindBuildPart: Procedure Expose XRc g.
                             /* -------------------------------------*\
                                Get name of start part and convert the
                                 '\' characters to '/' for TC Queries!
                             \*--------------------------------------*/
 Parse Arg startpart .
 startpart = Translate(startpart,"/","\")
                             /* -------------------------------------*\
                                We better clean up the REXX Queue first
                             \*--------------------------------------*/
 Do Queued(); Pull . ; End;
```

Figure 37 (Part 7 of 14). TWI_XBT.CMD

```
                              /* ---------------------------------------*\
                                 Initialize level and partlist
                              \*---------------------------------------*/
     buildpart = ""
     level = 0
     partlist.       = "0"
     partlist.level.0 = "1"
     partlist.level.1 = startpart

     /* ---------------------------------------------------------------*\
        Now start stepping up the build tree level by level...
     \*---------------------------------------------------------------*/
     Do level = 0 to 9
       partindex = 0
       nextlevel = level+1
       Do i = 1 to (partlist.level.0)
         mypart = partlist.level.i
         If (g.trace?) Then Do
           Say "Looking at part" mypart "now ..."
           End
                              /* ---------------------------------------*\
                                 Check if Part is build part ...
                              \*---------------------------------------*/
        "TEAMC Report -raw -view PartView",
                    "-where ""nuPathName = '"mypart"'""",
                    "-release" g.TC_RELEASE,
                    "-workarea" g.TC_WORKAREA "| RXQueue"

         builder1 = ""
         If (Queued())
           Then Parse Pull . "|" . "|" . "|" . "|" . "|",
                           . "|" . "|" . "|" . "|" . "|",
                           . "|" . "|" . "|" . "|" . "|",
                           . "|" . "|" . "|" . "|" builder1 "|" .

         Do Queued()          /* We only expect one line on the queue, */
           Pull . ; End       /*     ... but just be sure !            */
                              /* ---------------------------------------*\
                                 If its a build part, we are done !
                              \*---------------------------------------*/
         If (builder1 <> "") Then do
           buildpart = mypart
           Leave level /* loop */
           End
```

Figure 37 (Part 8 of 14). TWI_XBT.CMD

```
/* --------------------------------------------------------------*\
   It's no build part, so look for any parent parts now..
\*---------------------------------------------------------------*/
Else Do
  If (g.trace?) Then Do
    Say "   ... was not a builder part !"
    End

 "TEAMC Part -view" mypart ,
           "-long -release" g.TC_RELEASE,
           "-workarea" g.TC_WORKAREA "| RXQueue"
                        /* -------------------------------------*\
                           Save everything in a stem, we need the
                            REXX Queue within the loop again!
                        \*--------------------------------------*/
  QSaved.=""
  QSize  = Queued()
  Do QLine = 1 to QSize
    Parse Pull Qsaved.QLine
    End QLine

  InBuildRelationshipSection = 0

  Do QLine = 1 to Qsize

    If (QSaved.QLine = "build relationships:") Then Do
      InBuildRelationshipSection = 1
      Iterate QLine /* loop */
      End

    If (\InBuildRelationshipSection)
      Then Iterate Qline /* loop */

    If (QSaved.QLine = "versions:")
      Then Leave QLine /* loop */
```

Figure 37 (Part 9 of 14). TWI_XBT.CMD

```
         /* ---------------------------------------------------------*\
            So let's scan the build relationship lines for build parts
         \*----------------------------------------------------------*/
         Parse Var QSaved.QLine  partname parttype relation .
                             /*--------------------------------------*\
                                Put potential candidates into partlist
                                 for inspection in the next level
                             \*--------------------------------------*/
         If (relation = "OutputOf"),
          | (relation = "DependsOn") Then Do
           partname = Strip(partname)
           If (g.btree.partname.InList? = 1) Then Do
             buildpart = partname
             Leave level /* loop */
             End
           partindex = partindex + 1
           partlist.nextlevel.partindex = partname
           If (g.trace?) Then Do
             Say "   ... saved for next iteration:" partname
             End
           End
         End qline /* loop */
       End /* If (builder2 = "") Else */
     End i
     partlist.nextlevel.0 = partindex
   End level /* loop */

FBP_Return:

 If (buildpart = "")
   Then Say ">>> Could no find a build part for:" startpart
   Else Say ">>> Using build part:" buildpart

 Return buildpart

/* ---------------------------------------------------------------------*\
   Stop processing issuing an error message and giving back an Rc
\*----------------------------------------------------------------------*/
Abort: Procedure Expose XRc g. stem.
 Parse Arg retcode message
 Call ErrMsg(message)
 Call Done(retcode)
 Return
```

Figure 37 (Part 10 of 14). TWI_XBT.CMD

```
/* -------------------------------------------------------------------*\
 ErrMsg/InfMsg:

   Issue an Error or Informational message
    - Messages are preceeded by the program name
    - Messages are formatted to fit into a 80 Column line
    - Special formatting characters supported:
         ¬n = Line Break and set line indentation level to n (0=default)
           = Required Blank
    - ERROR Messages go to SDTERR
\*-------------------------------------------------------------------*/
ErrMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

InfMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

WrtMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring

 nam  = g.myname": "
 nams = Copies(" ",Length(nam))
 max  = 75-(Length(nam))
                             /* ---------------------------------------*\
                                Format the text lines
                             \*---------------------------------------*/
 n = 1;
 c = 0;
 msgl. = "";
 indent = 0;
 msgl.in.1 = indent;

 Do i = 1 While (msgstring <> "")
   Parse VAR  msgstring  w msgstring
   c = c + Length(w) +1
   Select
```
Figure 37 (Part 11 of 14). TWI_XBT.CMD

```
                             /* ---------------------------------------*\
                                Handle Line break and indention
                             \*----------------------------------------*/
        When (Left(w,1) = "¬" /* Indention */) Then Do
          indent = Substr(w,2)
          If (\DataType(indent,"W")) Then indent = 0
          c = 0
          n = n + 1
          msgl.in.n = indent
          End
                             /* ---------------------------------------*\
                                Compose the lines..
                             \*----------------------------------------*/
        Otherwise Do
          If (c <= max) Then Do
            msgl.n = msgl.n w
            End
          Else Do
            n = n + 1
            c = Length(w) +1
            msgl.n = w
            msgl.in.n = indent
            End
          End /* Otherwise */
        End /* Select */
      End i
                             /* ---------------------------------------*\
                                Write the lines ...
                                  and handle the required blanks
                             \*----------------------------------------*/
    Say
    Say nam || Copies(" ",msgl.in.1) || Translate(Strip(msgl.1)," ","")
    Do i = 2 to n
      Say nams || Copies(" ",msgl.in.i) || Translate(Strip(msgl.i)," ","")
      End i

    Return
```

Figure 37 (Part 12 of 14). TWI_XBT.CMD

```
/* ===================================================================*\
 * Here we catch all the errors ...
\*====================================================================*/
ERROR:
FAILURE:
NOTREADY: Procedure Expose sigl Rc XRc g. stem.
NOVALUE:
SYNTAX:

 Parse Upper Source myenv mycall mypname .
 Parse Value FileSpec("NAME",mypname) With myname "." .

 traptype = Condition("C")
 calltype = Condition("I")

 Say "REXX trap" calltype "on" traptype "in" mypname ":"

 Say "--->" sigl":" Strip(SourceLine(sigl))
```

Figure 37 (Part 13 of 14). TWI_XBT.CMD

```
                                           /* ----------------------------*\
                                              Trap specific processing
                                           \*-----------------------------*/
     Select
      When (traptype = "FAILURE") | (traptype = "ERROR") then do
        Say "Command   :" Condition("D")
        Say "Rc was    :" Rc
        end
      When traptype = "HALT" then do
        Say "Program has been interrupted by an external event!"
        end
      When traptype = "NOVALUE" then do
        Say "Variable  :" Condition("D")
        end
      When traptype = "SYNTAX" then do
        Say "Errortype :" ErrorText(Rc)
        end
      When traptype = "NOTREADY" then do
        Say "Stream    :" Condition("D")
        end
      Otherwise do
        Say "Unknown REXX trap condition >"traptype"<"
        Say "Trap Description :" Condition("D")
        end
      end /* Select */

     Call BEEP 1000,100

     Say "Hit any Key to continue ..."
     Pull .

     If (calltype = "SIGNAL") Then Exit 998
                              Else Return 999
```

Figure 37 (Part 14 of 14). TWI_XBT.CMD

Creating a Project Template

TWI_GPT: Generate Project Template

There are two ways to create and maintain a WorkFrame project setup:

- By using the WorkFrame GUI

  or
- By using the WorkFrame Project Smarts interface.

If you already have a good base, and only a few modifications are necessary, the GUI may be suitable for maintaining a project setup. But if you have to create a large number of actions frequently (as we had to), or your setup has to fit into different *inherited* environments and thus should have different action priorities, using the Project Smarts interface from a REXX program might be much more efficient.

Unfortunately there are some things that cannot be done from REXX, such as setting the project or file options for actions that use the default action support DLL (IWFOPT.DLL). Even if this final step must be done manually through the GUI, however, building the basic project setup using a REXX program will be easier and faster than using the GUI for every modification.

We used the REXX routine in Figure 38 on page 167 to create the base for the TWI_PRJ project file.

```
/* *******************************************************************
  TeamConnection / WorkFrame Integration

  File ..:  TWI_GPT.cmd
  Project:  Generate WF Project template (Actions) for the functions
            contained in the TWI_Do.Cmd script. The template project
            will be put into the VA C++ \smarts\projects directory.

  Author :  L. Sparmann  (SPARMANN @ BOEVM2)
  Owner .:  L. Sparmann  (SPARMANN @ BOEVM2)

  Description :

   (For further information, please see help!)

  Version and Change History:
  --------------------------*/
  version = "0.01 dated: ??.Mar.96  by: L. Sparmann"
  reason  = "Initial version ..."
/*
  version = "0.00 dated: 05.Mar.96  by: L. Sparmann"
  reason  = "Copied from C20_MON"

*********************************************************************/
 Address CMD
"@ECHO OFF"


                                       /* ----------------------------*\
                                         Setup for error trapping
                                       \*-----------------------------*/

 Signal on  NOVALUE  ; Signal on  SYNTAX ;           /* SIGNAL only ! */
 Call   on  FAILURE  ; Call   on  ERROR  ;           /* SIGNAL & CALL */
 Call   on  NOTREADY ; Signal on  HALT   ;           /* SIGNAL & CALL */

 /* ----------------------------------------------------------------*\
    Set up the global variables...
 \*-----------------------------------------------------------------*/
 XRc = 0
```

Figure 38 (Part 1 of 17). TWI_GPT.CMD

```
/* ----------------------------------------------------------------*\
   Basic setup ...
\*-----------------------------------------------------------------*/
g.            = "?"                 /* All my global variables ... */
                                    /*-----------------------------*/
g.step        = 0                   /* STEP and LASTSTEP are used  */
g.laststep    = 11                  /*   to indicate the progress. */
                                    /*-----------------------------*/
g.exitcmds    = ""                  /* Cmds to be executed on Exit */
                            /* ---------------------------------------*\
                               Setup WF specific Variables & Constants
                            \*----------------------------------------*/
stem. = "?"
stem.undefined = stem.undefined
                            /* ---------------------------------------*\
                               Get the call environment
                            \*----------------------------------------*/
Parse Upper Source g.myenv g.mycall g.mypname .
Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
g.mypath  = FileSpec("PATH" ,g.mypname)
g.mydrive = FileSpec("DRIVE",g.mypname)
/* ----------------------------------------------------------------*\
   We switch on monitoring as early as possible for debug reasons !
\*-----------------------------------------------------------------*/
XRc = IwfOpenConsole("STEM");
If (XRc <> 0) Then Do
  Call RxMessageBox "IwfOpenConsole failed, Rc was:" XRC,
                    ,g.myname":",
                    ,"ENTER",
                    ,"EXCLAMATION"
  Call Done(XRc)
  End
                            /* ---------------------------------------*\
                               Initial Monitor message
                            \*----------------------------------------*/
Call UpdateStatus("Initializing ...")
                            /* ---------------------------------------*\
                               Load OS/2 built-In Utilities
                            \*----------------------------------------*/
If (RxFuncQuery("SysLoadFuncs")) Then Do
  Call RxFuncAdd 'SysLoadFuncs','RexxUtil','SysLoadFuncs'
  Call SysLoadFuncs
  End
```

Figure 38 (Part 2 of 17). TWI_GPT.CMD

```
                              /* --------------------------------------*\
                                 Load support for Project tools setup
                              \*---------------------------------------*/
     If (RxFuncQuery("IwfEnvPrfLoadFuncs")) Then Do
       Call RxFuncAdd "IwfEnvPrfLoadFuncs",'IWFPAPI',"IwfRxEnvPrfLoadFuncs"
       Call IwfEnvPrfLoadFuncs
       Call IwfInitEnvPrfAPIs
       End
                              /* --------------------------------------*\
                                 It's good practice to clean up the
                                  REXX Queue in the beginnig
                              \*---------------------------------------*/
     Do Queued(); Pull .; End

     /* --------------------------------------------------------------------*\
        Get Arguments (as typed = mixed case) and provide help if desired
     \*---------------------------------------------------------------------*/
     Parse Arg allopts

     If (allopts = "?") Then Do
       Call InfMsg("¬ Syntax:" g.myname "<options>",
                   "¬",
                   "¬ Additional output for problem determination will be",
                     "produced, if the TRACE option is specified with this",
                     "command.")
       Call Done(100)
       End

     /* --------------------------------------------------------------------*\
        Look for debug or trace setings
     \*---------------------------------------------------------------------*/
     If (POS("TRACE",Translate(allopts)) = 0)
       Then g.trace? = 0
       Else Do
            g.trace? = 1
            Call Value "TWI_TRACE","ON","OS2ENVIRONMENT"
            End

    /* ---------------------------------------------------------------------*\
       Start processing now...
    \*----------------------------------------------------------------------*/
     /* --------------------------------------------------------------------*\
        Wee hook into the VA C++ projects smarts
     \*---------------------------------------------------------------------*/
     Call UpdateStatus("Searching for Projects Smarts folder...")
```

Figure 38 (Part 3 of 17). TWI_GPT.CMD

```
                             /* --------------------------------------*\
                                Find location of template project
                             \*---------------------------------------*/
     TemplatePath = g.mydrive||Strip(g.mypath,"Trailing","\")
     If (TemplatePath = "")
       Then Call Abort(99 "Error obtaining Project Smarts template path.");
                             /* --------------------------------------*\
                                Preset interface variables and start
                                  Location dialog ...
                             \*---------------------------------------*/
     stem.pszTargetProject   = "TWI_GPT"
     stem.pszTargetDirectory = "D:\not-applicable"
     stem.pszTargetFolder    = TemplatePath
                             /* --------------------------------------*\
                                For problem determination only ...
                             \*---------------------------------------*/
     If (g.trace?) Then Do
       Call InfMsg("¬ Target Information:"  ,
                   "¬3   TargetProject:"   stem.pszTargetProject,
                   "¬3   TargetDirectory:" stem.pszTargetDirectory,
                   "¬3   TargetFolder:"    stem.pszTargetFolder,
                   "")
       End

     /* --------------------------------------------------------------*\
        Now create the WF project template ...
     \*---------------------------------------------------------------*/
     Call UpdateStatus("The project will be created in "||stem.pszTargetFolder)
```

Figure 38 (Part 4 of 17). TWI_GPT.CMD

```
                                  /* --------------------------------------*\
                                     Create the project now...
                                  \*---------------------------------------*/
     setup_string = ″RUNPROMPT=FALSE;″ ,
                 || ″RUNMONITORED=FALSE;″ ,
                 || ″FILTER=*.*;″ ,
                 || ″PAMORDER=IWFBPAM;″ ,
                 || ″MONAUTOSCROLL=YES;″ ,
                 || ″MONAUTOERASE=NO;″ ,
                 || ″MONDISPLAYONSTART=YES;″ ,
                 || ″MONHIDECOMPLETION=NO;″,
                 || ″″

     If (g.trace?) Then Do
       Call InfMsg(″¬ Object Setup String: ¬3″ setup_string)
       End

     CORc = SysCreateObject(″IWFProject″,,
                            stem.pszTargetProject ,,
                            stem.pszTargetFolder ,,
                            setup_string,,
                            ″REPLACE″)

     If (g.trace?) Then Do
       Call InfMsg(″SysCreateObject returned:″ CORc)
       End

     If (CORc = 0)
       Then Call Abort Rc ″Could not create Object:″ stem.pszTargetProject,
                        ″Return code from SysCreateObject was″ CORc
```

Figure 38 (Part 5 of 17). TWI_GPT.CMD

```
/* -----------------------------------------------------------------*\
   Create Tools Setup (Actions...)
\*------------------------------------------------------------------*/
Call UpdateStatus(”Creating File Actions ...”)

stem.pszProject = stem.pszTargetFolder”\”stem.pszTargetProject

stem.pszCommand       = ”TWI_DO.CMD”
stem.pszDllName       = ”IWFOPT”
stem.pszHelpCmd       = ”VIEW”
stem.pszHelpTopic     = ”none”
stem.pszPam           = ”IWFBPAM”

stem.pszSrcMask       = ”*.*”
stem.pszTgtMask       = ” ”
```

Figure 38 (Part 6 of 17). TWI_GPT.CMD

```
                                  /* --------------------------------------*\
                                     File actions
                                  \*---------------------------------------*/
     /*              Action Class:    Action Name:            Source Mask:
                     --------------   --------------------     -----------
                     IWFOPT:  Key Pri  FPB:      DFWM:     POT  Parameters:
                     -------  --- ---  -------   -------   ---  -----------------*/
     Call AddAction "Edit          ","TC Edit              ","Editable  ",,
                    "EDIT   "," ",71 ,"FILE   ","Default","x--","EDIT       %a %z"
     Call AddAction "TeamConnection","TC Touch             ","NotTcSaved",,
                    "DEFAULT"," ",63 ,"FILE   ","Monitor","x--","TOUCH      %a %z"
     Call AddAction "TeamConnection","TC CheckIn (Forced)  ","Editable  ",,
                    "DEFAULT"," ",68 ,"FILE   ","Monitor","x--","CHECKIN+   %a %z"
     Call AddAction "TeamConnection","TC CheckIn           ","Editable  ",,
                    "DEFAULT"," ",67 ,"FILE   ","Monitor","x--","CHECKIN    %a %z"

     Call UpdateStatus("Creating File Actions ... ...")

     Call AddAction "TeamConnection","TC CheckOut          ","Editable  ",,
                    "DEFAULT"," ",66 ,"FILE   ","Monitor","x--","CHECKOUT   %a %z"
     Call AddAction "TeamConnection","TC Lock              ","Editable  ",,
                    "DEFAULT"," ",65 ,"FILE   ","Monitor","x--","LOCK       %a %z"
     Call AddAction "TeamConnection","TC UnLock            ","Editable  ",,
                    "DEFAULT"," ",64 ,"FILE   ","Monitor","x--","UNLOCK     %a %z"
     Call AddAction "TeamConnection","TC Extract Part(s)   ","NotTcSaved",,
                    "DEFAULT"," ",62 ,"FILE   ","Monitor","x--","EXTRACT    %a %z"

     Call UpdateStatus("Creating File Actions ... ... ...")

     Call AddAction "View          ","TC View Build Messages","NotTcSaved",,
                    "DEFAULT"," ",66 ,"FILE   ","Monitor","x--","VIEWBLDMSG %a %z"
     Call AddAction "Build         ","TC Build Part        ","NotTcSaved",,
                    "DEFAULT"," ",67 ,"FILE   ","Monitor","x--","BLDPNORM   %f   "
     Call AddAction "Build         ","TC Build Part Forced  ","NotTcSaved",,
                    "DEFAULT"," ",66 ,"FILE   ","Monitor","x--","BLDPFORCE  %f   "
```

Figure 38 (Part 7 of 17). TWI_GPT.CMD

```
                              /* --------------------------------------*\
                                 Project actions
                              \*---------------------------------------*/
     Call UpdateStatus("Creating Project Actions ...")

     Call AddAction "Build         ","TC Build Target       ","NotTcSaved",,
                    "DEFAULT"," ",97 ,"PROJECT","Monitor","x-x","BLDTNORM        "
     Call AddAction "Build         ","TC Build Target Forced","NotTcSaved",,
                    "DEFAULT"," ",96 ,"PROJECT","Monitor","x--","BLDTFORCE       "

     Call UpdateStatus("Creating Project Actions ... ...")

     Call AddAction "TeamConnection","TC Freeze WorkArea    ",            ,,
                    "DEFAULT"," ",67 ,"PROJECT","Monitor","x--","WAFREEZE        "
     Call AddAction "TeamConnection","TC Task List          ",            ,,
                    "DEFAULT"," ",66 ,"PROJECT","Default","x-x","TCTASK          "
     Call AddAction "TeamConnection","TC Show WorkArea      ",            ,,
                    "DEFAULT"," ",65 ,"PROJECT","Default","x--","WASHOW          "

     /* ------------------------------------------------------------------*\
        Register useful Type Classes ...
     \*-------------------------------------------------------------------*/
     Call UpdateStatus("Registering useful Type Classes ...")

     Call RegisterIWFTypeClasses

     /* ------------------------------------------------------------------*\
        Create Types ...
     \*-------------------------------------------------------------------*/
     Call UpdateStatus("Creating Types ...")

     Call AddType "TcSaved        ", "FileMask           ", "*.$*"
     Call AddType "NotTcSaved     ", "NOT IN FileMask    ", "*.$*"

     Call Done(0)
```

Figure 38 (Part 8 of 17). TWI_GPT.CMD

```
/* -------------------------------------------------------------------*\
   Jump to this point to exit the program
\*--------------------------------------------------------------------*/
Done:

 Arg XRc

 Call UpdateStatus(″Generation of Workframe Projects completed, (Rc=″XRc″)″)

 Call RxMessageBox ″The Creation of the Teamconnection/Workframe″,
                   ″integration project template has ended.″,
                   ″Return Code was″ XRc ″!″,
                  ,g.myname″:″,
                  ,″OK″,
                  ,″INFORMATION″
                             /* -------------------------------------*\
                                This should better be reset in case
                                 the program was interrupted by CTL-C
                             \*--------------------------------------*/
HALT:
                             /* -------------------------------------*\
                                Close Installation Console ...
                             \*--------------------------------------*/
 dyRc = IwfTermEnvPrfAPIs()
 dyRc = IwfEnvPrfDropFuncs()
 dyRc = IwfCloseConsole(″STEM″);

 Exit XRc

/* ==================================================================*\
 * Start of local subroutines ...
\*===================================================================*/
```

Figure 38 (Part 9 of 17). TWI_GPT.CMD

```
/* -------------------------------------------------------------------*\
   Update the status line and write a log entry indicating the current
   step of the project setup...
\*--------------------------------------------------------------------*/
UpdateStatus: Procedure Expose XRc g. stem.

 Parse Arg text

 g.step = g.step + 1
 If (g.step > g.laststep) Then Do
   Call ErrMsg("too many steps, please update G.LASTSTEP to:" g.step "!")
   stem.usPercent = g.laststep
   End
 Else stem.usPercent = (100 / g.laststep * g.step) % 1

 rc = IwfUpdateConsoleProgress("STEM");
 If (Rc <> 0) Then Do
   Call ErrMsg("IwfUpdateConsoleStatus failed, Rc was:" Rc)
   End

 Call InfMsg("("g.step")" text)
 stem.pszStatusText = text
 rc = IwfUpdateConsoleStatus("STEM");
 If (Rc <> 0) Then Do
   Call ErrMsg("IwfUpdateConsoleStatus failed, Rc was:" Rc)
   End
 Return

/* -------------------------------------------------------------------*\
   Register a useful set of IWF Type Classes
\*--------------------------------------------------------------------*/
RegisterIWFTypeClasses:
 stem.pszDllName = "IWFTYPES"

 Call RegisterTypeClass "FileMask"            , "IWFFileMask"
 Call RegisterTypeClass "Logical AND"         , "IWFLogAND"
 Call RegisterTypeClass "Logical OR"          , "IWFLogOR"
 Call RegisterTypeClass "NOT IN FileMask"     , "IWFNotInFileMask"
 Call RegisterTypeClass "NOT IN Logical AND"  , "IWFNotInLogAND"
 Call RegisterTypeClass "NOT IN Logical OR"   , "IWFNotInLogOR"

 Return
```

Figure 38 (Part 10 of 17). TWI_GPT.CMD

```
/* ------------------------------------------------------------------*\
   Register a IWF Type Class
\*-------------------------------------------------------------------*/
RegisterTypeClass:

 Parse Arg class , entry .

 stem.pszClass      = Strip(class)
 stem.pszEntryPoint = Strip(entry)

 Rc = IwfRegisterTypeClass("STEM")

 If (Rc <> 0)
   Then Call Abort Rc "Could not register Class:" stem.pszClass "(Rc="Rc")"

 Return

/* ------------------------------------------------------------------*\
   Add IWF Type
\*-------------------------------------------------------------------*/
AddType:

 Parse Arg type , class , values

 stem.pszName          = Strip(type)
 stem.pszClass         = Strip(class)
 stem.pszValue         = Strip(values)

 Rc = IwfAddType("STEM")

 If (Rc <> 0)
   Then Call Abort Rc "Could not Add Type:" stem.pszName "(Rc="Rc")"

 Return
```

Figure 38 (Part 11 of 17). TWI_GPT.CMD

```
/* -------------------------------------------------------------------*\
   Add Action to tools setup:

       ActClass = Action Class Name
       ActName  = Action Name
       IwfOpt   = Entry point of IWFOPT to be used
       Key      = Accelerator Key (character + Ctrl + SHIFT)
       Pri      = Priority
       fpb      = Action applies to: File, Project, Both
       dfwm     = Session type: Default, Fullscreen, Window, Monitored
       pot      = If (p='-') then do not add to Project Menu
                  If (o='-') then do not add to Options Menu
                  If (t='-') then do not add to Toolbar
\*-------------------------------------------------------------------*/
AddAction: Procedure Expose g. stem.

 Parse Arg ActClass , ActName , SrcMask , IwfOpt , Key , Priority . ,,
           fpb . , dfwm . , pot , Parms

 stem.pszActionClass   = Strip(ActClass)
 stem.pszActionName    = Strip(ActName)
 stem.pszSrcMask       = Strip(SrcMask)
 stem.pszDllEntryName  = Strip(IwfOpt)
 stem.pszucAccelKey    = Key
 stem.pszPriority      = Priority
 stem.pszucActionScope = fpb
 stem.pszucRunMode     = Left(dfwm,1)

 If (Substr(pot,1,1) = "-")
   Then stem.pszfPrjMenu      = "F" /* Project Menu: F=false */
   Else stem.pszfPrjMenu      = "T" /* Project Menu: T=true  */

 If (Substr(pot,2,1) = "-")
   Then stem.pszfOptMenu      = "F" /* Options Menu: F=false */
   Else stem.pszfOptMenu      = "T" /* Options Menu: T=true  */

 If (Substr(pot,3,1) = "-")
   Then stem.pszfTBMenu       = "F" /* Tool Bar    : F=false */
   Else stem.pszfTBMenu       = "T" /* Tool Bar    : T=true  */

 If (stem.pszSrcMask = "")
   Then stem.pszSrcMask = "*.*"
```

Figure 38 (Part 12 of 17). TWI_GPT.CMD

```
/***
 stem.pszCommand       = ″TWI_″Word(parms,1)″.cmd″
 ***/

 Rc = IwfAddAction(″STEM″)

 If (Rc <> 0)
   Then Call Abort Rc ″Could not Add Action:″ stem.pszActionName ″(Rc=″Rc″)″

 Return

/* ------------------------------------------------------------------*\
   Confirm Cancellation
\*-------------------------------------------------------------------*/
Cancel:
 rc = RxMessageBox(″Do you really want to cancel?″,
                  ,g.myname″:″,
                  ,″YESNO″,
                  ,″QUERY″)
 If (rc = 6 /*YES*/)
   Then Call Done(8)
 Return;

/* ------------------------------------------------------------------*\
   Stop processing issuing an error message and giving back an Rc
\*-------------------------------------------------------------------*/
Abort: Procedure Expose XRc g. stem.
 Parse Arg retcode message
 Call ErrMsg(message)
 Call Done(retcode)
 Return
```

Figure 38 (Part 13 of 17). TWI_GPT.CMD

```
/* ------------------------------------------------------------------*\
 ErrMsg/InfMsg:

   Issue an Error or Informational message
    - Messages are preceeded by the program name
    - Messages are formatted to fit into a 80 Column line
    - Special formatting characters supported:
         ¬n = Line Break and set line indentation level to n (0=default)
           = Required Blank
    - ERROR Messages go to SDTERR
\*-------------------------------------------------------------------*/
ErrMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

InfMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring
 Call WrtMsg(msgstring)
 Return

WrtMsg: Procedure Expose XRc g. stem.
 Parse Arg msgstring

 nam  = g.myname": "
 nams = Copies(" ",Length(nam))
 max  = 75-(Length(nam))
                             /* ---------------------------------------*\
                                Format the text lines
                             \*----------------------------------------*/
 n = 1;
 c = 0;
 msgl. = "";
 indent = 0;
 msgl.in.1 = indent;

 Do i = 1 While (msgstring <> "")
   Parse VAR  msgstring  w msgstring
   c = c + Length(w) +1
   Select
```
Figure 38 (Part 14 of 17). TWI_GPT.CMD

```
                          /* --------------------------------------*\
                             Handle Line break and indention
                          \*---------------------------------------*/
    When (Left(w,1) = "¬" /* Indention */) Then Do
      indent = Substr(w,2)
      If (\DataType(indent,"W")) Then indent = 0
      c = 0
      n = n + 1
      msgl.in.n = indent
      End
                          /* --------------------------------------*\
                             Compose the lines..
                          \*---------------------------------------*/
    Otherwise Do
      If (c <= max) Then Do
        msgl.n = msgl.n w
        End
      Else Do
        n = n + 1
        c = Length(w) +1
        msgl.n = w
        msgl.in.n = indent
        End
      End /* Otherwise */
    End /* Select */
  End i
                          /* --------------------------------------*\
                             Write the lines ...
                               and handle the required blanks
                          \*---------------------------------------*/
Say
Say nam || Copies(" ",msgl.in.1) || Translate(Strip(msgl.1)," ","")
Do i = 2 to n
  Say nams || Copies(" ",msgl.in.i) || Translate(Strip(msgl.i)," ","")
  End i

Return
```

Figure 38 (Part 15 of 17). TWI_GPT.CMD

```
/* ==================================================================*\
 * Here we catch all the errors ...
\*==================================================================*/
ERROR:
FAILURE:
NOTREADY: Procedure Expose sigl Rc XRc g. stem.
NOVALUE:
SYNTAX:

 Parse Upper Source myenv mycall mypname .
 Parse Value FileSpec(”NAME”,mypname) With myname ”.” .

 traptype = Condition(”C”)
 calltype = Condition(”I”)

 Say ”REXX trap” calltype ”on” traptype ”in” mypname ”:”

 Say ”--->” sigl”:” Strip(SourceLine(sigl))
```

Figure 38 (Part 16 of 17). TWI_GPT.CMD

```
                                           /* ----------------------------*\
                                              Trap specific processing
                                           \*-----------------------------*/
     Select
      When (traptype = ″FAILURE″) | (traptype = ″ERROR″) then do
        Say ″Command   :″ Condition(″D″)
        Say ″Rc was    :″ Rc
        end
      When traptype = ″HALT″ then do
        Say ″Program has been interrupted by an external event!″
        end
      When traptype = ″NOVALUE″ then do
        Say ″Variable  :″ Condition(″D″)
        end
      When traptype = ″SYNTAX″ then do
        Say ″Errortype :″ ErrorText(Rc)
        end
      When traptype = ″NOTREADY″ then do
        Say ″Stream    :″ Condition(″D″)
        end
      Otherwise do
        Say ″Unknown REXX trap condition >″traptype″<″
        Say ″Trap Description :″ Condition(″D″)
        end
      end /* Select */

     Call BEEP 1000,100

     If (calltype = ″SIGNAL″) Then Exit 998
                              Else Return 999
```

Figure 38 (Part 17 of 17). TWI_GPT.CMD

Other Useful REXX Programs

A REXX Message Box for non-Presentation Manager Environments

The REXX utility functions (REXXUTIL) provide the RcMessageBox function, which enables you to pop up a message box from a REXX program if you want to inform the user about some event that happened in a background task. Unfortunately, this function only works when invoked from a Presentation Manager (PM) application.

The MSGBOX.CMD (Figure 39 on page 185) and MSGBOX1.CMD (Figure 40 on page 188) files provide a way of using the RxMessageBox function from a non-PM environment. MSGBOX has the same call interface as RxMessageBox, but it returns the name of the button instead of the return code.

Examples:

```
button = MsgBox(″Do you want to check in″ part, /* Text    */
               ,g.myname″:″ g.U_function,       /* Title   */
               ,″YesNo″,                        /* Buttons */
               ,″Query″)                        /* Icon    */

If (button = ″YES″) Then Do
  ...
  End
```

```
/* -------------------------------------------------------------------*\
   MsgBox.CMD

   Together with it's counterpart, MsgBox1.CMD, this program provides
   a way of using the RxMessageBox function in a non PM environment.

   MsgBox returns the button that was pressed by the user. The values
   returned will be one of the list below:

          OK, CANCEL, ABORT, RETRY, IGNORE, YES, NO, ENTER

   The string '???' will be returned if an undefined value is returned
   by the RxMessageBox Function !

   L. Sparmann  (SPARMANN at BOEVM2)
\*-------------------------------------------------------------------*/

 Address 'CMD'
 Signal On Halt

"@ECHO OFF"
                             /* --------------------------------------*\
                                Get the call environment
                             \*--------------------------------------*/
 Parse Upper Source g.myenv g.mycall g.mypname .
 Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
                             /* --------------------------------------*\
                                Create a REXX queue for communication
                             \*--------------------------------------*/
 MBoxQ = RXQueue("CREATE")
 SessQ = RxQueue("SET",MBoxQ)
 Do Queued(); Pull . ; End;  /* Clean up the queue ! */

 /* ------------------------------------------------------------------*\
    Now let's do the job...
 \*------------------------------------------------------------------*/
```

Figure 39 (Part 1 of 3). MSGBOX.CMD

```
                              /* --------------------------------------*\
                                 Get arguments (case sensitive)
                              \*---------------------------------------*/
  Parse Arg parm1 , parm2 , parm3 , parm4
  If (parm2 = ””)
    Then Parse Var parm1  parm1 ′,′  parm2 ′,′  parm3 ′,′  parm4

  parm1 = Strip(parm1)
  parm2 = Strip(parm2)
  parm3 = Strip(parm3)
  parm4 = Strip(parm4)
                              /* --------------------------------------*\
                                 Starting my PM counterpart now ...
                              \*---------------------------------------*/
 ”START /PM CMD /K” g.myname”1” MBoxQ
                              /* --------------------------------------*\
                                 Wait until couterpart indicates that
                                   it is ready by sending Id of MBoxQ1
                              \*---------------------------------------*/
  Do infinite_loop=1
    If (Queued() = 1) Then Do
      Parse Pull MBoxQ1
      Leave infinite_loop
      End
      Else Call SysSleep 1
    End infinite_loop
                              /* --------------------------------------*\
                                 Send parameters over MBoxQ1 ...
                              \*---------------------------------------*/
  Call  RxQueue ”SET”,MBoxQ1
  Do Queued(); Pull . ; End;  /* Clean it up first ! */
  Queue parm1
  Queue parm2
  Queue parm3
  Queue parm4
```

Figure 39 (Part 2 of 3). MSGBOX.CMD

```
                                /* -------------------------------------*\
                                   Wait for users response on MBoxQ ...
                                \*--------------------------------------*/
     Call  RxQueue "SET",MBoxQ
     Do infinite_loop=1
       If (Queued() = 1) Then Do
         Parse Pull MBXRc
         Leave infinite_loop
         End
       Else Call SysSleep 1
       End infinite_loop
                                /* -------------------------------------*\
                                   Translate MBXRc to button
                                \*--------------------------------------*/
     Select
       When (MBXRc = 1) Then button = "OK"
       When (MBXRc = 2) Then button = "CANCEL"
       When (MBXRc = 3) Then button = "ABORT"
       When (MBXRc = 4) Then button = "RETRY"
       When (MBXRc = 5) Then button = "IGNORE"
       When (MBXRc = 6) Then button = "YES"
       When (MBXRc = 7) Then button = "NO"
       When (MBXRc = 8) Then button = "ENTER"
       Otherwise             button = "???"
       End

    /* ---------------------------------------------------------------*\
       We are done, clean up the environment !
    \*----------------------------------------------------------------*/
    Clean_Exit:
    Halt:

     dyRc = RXQueue("SET",SessQ)
     dyRc = RXQueue("DELETE",MBoxQ)

     If (g.mycall = "COMMAND")
       Then Say button
       Else Return button

     Exit XRc
```

Figure 39 (Part 3 of 3). MSGBOX.CMD

```
/* -------------------------------------------------------------------*\
   MsgBox1.CMD

   Is the counterpart to MsgBox.CMD providing the RxMessageBox
   functionality for a non PM environment...

   L. Sparmann  (SPARMANN at BOEVM2)
\*--------------------------------------------------------------------*/

 Address 'CMD'
 Signal On Halt

"@ECHO OFF"
                             /* --------------------------------------*\
                                Get the call environment
                             \*---------------------------------------*/
 g. = ""
 Parse Upper Source g.myenv g.mycall g.mypname .
 Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
                             /* --------------------------------------*\
                                Create a REXX queue to be used for
                                parameter passing...
                                Send name of this queue to the caller
                             \*---------------------------------------*/
 MBoxQ1 = RXQueue("CREATE")

 /* ------------------------------------------------------------------*\
    Let's do the job now ...
 \*-------------------------------------------------------------------*/
                             /* --------------------------------------*\
                                Get callers Queue Id
                             \*---------------------------------------*/
 Parse Arg MBoxQ
 Say g.myname": Got Queue Id:" MBoxQ
                             /* --------------------------------------*\
                                Send back own queue Id
                             \*---------------------------------------*/
 SessQ = RxQueue("SET",MBoxQ)
 Queue MBoxQ1
```

Figure 40 (Part 1 of 2). MSGBOX1.CMD

```
                              /* --------------------------------------*\
                                 Get prameters for RxMessageBox
                              \*---------------------------------------*/
 Call RxQueue "SET",MBoxQ1
 Do infinite_loop=1
   If (Queued() = 4) Then Do /* Start if everything is available ...  */
     Parse Pull parm1
     Parse Pull parm2
     Parse Pull parm3
     Parse Pull parm4
     Leave infinite_loop
     End
   Else Call SysSleep 1
   End infinite_loop
                              /* --------------------------------------*\
                                 Display Message Box now and return Rc
                              \*---------------------------------------*/
 MBXrc = RxMessageBox(parm1,parm2,parm3,parm4)

 Call RxQueue "SET",MBoxQ
 Queue MBXRc

/* -----------------------------------------------------------------*\
   We are done, don't forget to delete our queue !
\*------------------------------------------------------------------*/
Clean_Exit:
Halt:

 dyRc = RXQueue("SET",SessQ)
 dyRc = RXQueue("DELETE",MBoxQ1)

 Exit 0
```

Figure 40 (Part 2 of 2). MSGBOX1.CMD

LXSYNC: A Synchronous Way of Invoking LPEX

The powerful live parsing editor (LPEX) included in VisualAge for C++ does not provide a synchronous call/return interface. In fact, the editor itself stays in memory after the first invocation, and in all subsequent invocations the editor will return to its caller as soon as a file is loaded. Therefore the caller is not informed when the editing session ends.

Some actions (like TC Edit) require a *synchronous* invocation and have to know when the editing session ends. Therefore we devised a quick solution for a synchronous invocation, using RxQueue (Figure 41) and an editor macro (Figure 42 on page 193) that intercepts the `File` and `Quit` command processing of the editor.

```
/* -------------------------------------------------------------------*\
   LxSync.CMD

   Is the counterpart to LxSync.LX trying to simulate a synchronous
   invocation of an LPEX editor session. LxSync starts the LPEX Editor
   (EVFXLXPM) for ONE file, so that the macro LxSync.LX is invoked with
   the SETUP parameter and the name of a REXX Queue that is used for
   communication.

   LxSync.LX returns the information whether the user ended the LPEX
   session in a FILE or QUIT like manner followed by the filename.

   LxSync.CMD returns depending on the type of invocation:

     COMMAND  - Returncode
     FUNCTION - Returncode Filename

   A return code of 99 means that the user ended the editor session
   by QUITting without SAVEing

   L. Sparmann  (SPARMANN at BOEVM2)
\*--------------------------------------------------------------------*/

 Address 'CMD'
 Signal On Halt

"@ECHO OFF"
 XRc = 100
```

Figure 41 (Part 1 of 3). LXSYNC.CMD

```
                             /* ---------------------------------------*\
                                Get the call environment
                             \*----------------------------------------*/
  Parse Upper Source g.myenv g.mycall g.mypname .
  Parse Value FileSpec("NAME",g.mypname) With g.myname "." .
  g.mypath  = FileSpec("PATH" ,g.mypname)
  g.mydrive = FileSpec("DRIVE",g.mypname)
                             /* ---------------------------------------*\
                                Get arguments (case sensitive)
                             \*----------------------------------------*/
  Parse Arg allargs
  Parse Var allargs file . "/" allopts
  If (allopts <> "") Then allopts = "/"allopts
                             /* ---------------------------------------*\
                                OS/2 built-In Utilities
                             \*----------------------------------------*/
  If (RxFuncQuery("SysLoadFuncs")) Then Do
    Call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
    Call SysLoadFuncs
    End
                             /* ---------------------------------------*\
                                Create a REXX queue to communicate
                                  with the LPEX session an initially
                                  clean it up !
                             \*----------------------------------------*/
  LxSyncQ = RXQueue("CREATE")
  SesQ    = RxQueue("SET",LxSyncQ)  /* saves name of active REXX queue */
  Do Queued(); Pull . ; End;

  /* ---------------------------------------------------------------------*\
     Invoke LPEX so that my .LX counterpart gains control ...
       Note, that an additional /CM is passed to it as a parameter !
  \*----------------------------------------------------------------------*/
  Say g.myname" invoking:"
  Say "  START EVFXLXPM" file "/CM" g.myname".LX SETUP" LxSyncQ allopts

 "START EVFXLXPM" file "/CM" g.myname".LX SETUP" LxSyncQ allopts
```

Figure 41 (Part 2 of 3). LXSYNC.CMD

```
 /* ----------------------------------------------------------------*\
    Look if the LPEX editor session has put something on the queque
 \*-----------------------------------------------------------------*/
 Do infinite_loop=1
   Do q_loop=1 to Queued()
     Parse Pull message filename
     message = Translate(message)
     If (message = "CONNECTED") Then Do
       Iterate q_loop
       End
     If (message = "FILE") Then Do
       Xrc = 0
       Leave infinite_loop
       End
     If (message = "QUIT") Then Do
       Xrc = 99
       Leave infinite_loop
       End
     Say "Garbage received on Queue ("LxSyncQ"):" message filename
     End q_loop

   Call SysSleep 2
   End infinite_loop

/* -----------------------------------------------------------------*\
   We are done, clean up the environment !
\*------------------------------------------------------------------*/
Clean_Exit:
Halt:

 Call RXQueue "Set",SesQ
 dyRc = RXQueue("DELETE",LxSyncQ)

 If (g.mycall = "COMMAND") Then Do
   Say XRc filename
   Exit XRc
   End

 Return XRc filename
```

Figure 41 (Part 3 of 3). LXSYNC.CMD

```
/* -------------------------------------------------------------------*\
   LxSync.LX

   Is the counterpart to LxSync.CMD trying to simulate a synchronous
   invocation of an LPEX editor session. LxSyn intercepts the FILE ans
   QUIT commands to communicate the end of an editing session to the
   calling program (LxSyn.CMD).
   A REXX Queue is used for communication. The name of the queue as well
   as the proevious settings of the synonyms file & quit are remembered
   in LPEX synonyms starting with LXSYNC_.

   LxSync.LX returns the information whether the user ended the LPEX
   session in a FILE or QUIT like manner followed by the filename.

   L. Sparmann  (SPARMANN at BOEVM2)
\*-------------------------------------------------------------------*/
 Trace o

 Arg func LxSyncQ other_lpex_command

 Parse Upper Source g.myenv g.mycall g.mypname .
 Parse Value FileSpec(”NAME”,g.mypname) With g.myname ”.” .

                             /* --------------------------------------*\
                                Provide help on invalid calls
                             \*---------------------------------------*/
 If (func = ””) Then Do
   ”MSG Syntax:” g.myname ”·SETUP qname <macrocall> | FILE | QUIT“”
    Exit 100
    End
                             /* --------------------------------------*\
                                Estabish Synonyms for File & Quit...
                                ...and execute other macro if specified
                             \*---------------------------------------*/
 Select
   When (func = ”SETUP”) Then Do

    ”Extract name ”
     Say g.myname”:>>>” LxSyncQ ”on:” name

     SesQ = RxQueue(”SET”,LxSyncQ)
     Queue ”CONNECTED”                /* Signal successful connection */
     Call RxQueue ”SET”,SesQ

    ”Set Synonym.”g.myname”_Queue” LxSyncQ
```

Figure 42 (Part 1 of 3). LXSYNC.LX

```
                             /* --------------------------------------*\
                                Do not use LXN QQUIT in a workframe
                                 environment, they too are intercepting
                                 the end of the session setting a
                                 Synonym on QQUIT !!!
                             \*---------------------------------------*/
        ″Set SYNONYM.QUIT MULT ;″ g.myname″.LX QUIT ;″ /*LXN*/ ″QQUIT″
        ″Set SYNONYM.FILE MULT ;″ g.myname″.LX FILE ;″ /*LXN*/ ″QQUIT″

         If (other_lpex_command <> ″″) Then Do
           Say g.myname″:>>> Invoking:″
           Say ″    ″ other_lpex_command
           other_lpex_command
           End
         End
                             /* --------------------------------------*\
                                If QUIT was executed...
                                   ask if user wants to save the file
                                   ... and return to the caller !
                             \*---------------------------------------*/
       When (func = ″QUIT″) Then Do
           /* We do not look for the CHANGES parameter, because it might not
              be set if the user did not really hit enter before he quits! */

           action = RxMessageBox(″Do you want to save the changes?″, ,
                             g.myname″:″,″YESNO″,″QUESTION″)
           If (action = 6) /* YES */ Then do
            ″LXN SAVE″
             Call Send_Completion_Msg ″FILE″
             End
           Else Do
             Call Send_Completion_Msg func
             End
         End
                             /* --------------------------------------*\
                                Save the file and return to the caller.
                             \*---------------------------------------*/
       When (func = ″FILE″) Then Do
        ″LXN SAVE″
         Call Send_Completion_Msg func
         End
```

Figure 42 (Part 2 of 3). LXSYNC.LX

```
                              /* --------------------------------------*\
                                 Just in case ...
                              \*---------------------------------------*/
   Otherwise Do
    ”MSG” g,myname”: Unsupported function” func
     End
   End /* SELECT */

  Exit 0


/* -------------------------------------------------------------------*\
   Indicate completion to the caller ...

     SAVE - File has been saved by the user
     QUIT - User did not want to save the file
\*--------------------------------------------------------------------*/
Send_completion_msg:

 Arg cmsg .

”Extract name Synonym.”g.myname”_Queue into LxSyncQ”
                       Parse Var LxSyncQ  . LxSyncQ

 Say g.myname”:<<<” LxSyncQ cmsg name
 SesQ = RxQueue(”SET”,LxSyncQ)
 Queue cmsg name
 Call RxQueue ”SET”,SesQ

 Return
```

Figure 42 (Part 3 of 3). LXSYNC.LX

Glossary

This glossary defines the terms and abbreviations used in this book. If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

A

absolute path name. A directory or a part expressed as a sequence of directories followed by a part name beginning from the root directory.

access list. A set of objects that controls access to data. Each object consists of a component, a user, and the authority that the user is granted or is restricted from in that component. See also *authority* and *restricted authority*.

action. A task performed by the TeamConnection server and requested by a TeamConnection client. A TeamConnection action is the same as issuing one TeamConnection command.

agent. See *build agent*.

alternate version ID. In collision records, the name of a version of a driver, release, or work area where the conflicting version of a part is visible.

approval record. A status record on which an approver must give an opinion of the proposed part changes required to resolve a defect or implement a feature in a release.

approver. A user who has the authority to mark an approval record with accept, reject, or abstain within a specific release.

approver list. A list of user IDs attached to a release, representing the users who must review part changes that are required to resolve a defect or implement a feature in that release.

attribute. Information contained in a field that is accessible to the user. TeamConnection enables family administrators to customize defect, feature, user, and part tables by adding new attributes.

authority. The right to access development objects and perform TeamConnection commands. See also *access list*, *base authority*, *explicit authority*, *implicit authority*, and *restricted authority*.

B

base authority. The set of actions granted to a user when a user ID is created within a TeamConnection family. See also *authority*. Contrast with *implicit authority* and *explicit authority*

build. The process used to create applications within TeamConnection.

build agent. A program that handles access to persistent data on behalf of the build processor. Each build agent is connected to one and only one build processor, through a TCP/IP connection.

build cache. A directory that the build processor uses to enhance performance.

build dependent. A TeamConnection part that is needed for the compile operation to complete but will not be passed directly to the compiler. An example of this is an include file. See also *dependencies*.

builder. An object that can transform one set of TeamConnection parts into another by invoking tools such as compilers and linkers.

build event. An individual step in the build of an application, such as the compiling of `hello.c` into `hello.obj`.

build input. A TeamConnection part that will be used as input to the object being built.

build output. A TeamConnection part that will be generated as output from a build, such as an `.obj` or `.exe` file.

build pool. A group of build servers that resides in an environment. The environment in which several build servers operate. Typically, several servers are set up for each environment for which the enterprise develops applications.

build processor. A program that invokes tools, such as compilers and linkers, that construct an application. Each build processor is connected to one and only one build agent, through a TCP/IP connection. See also *build agent* and *build cache*.

build scope. A collection of build events that implement a specific build request. See also *build event*.

build script. An executable or command file that specifies the steps that should occur during a build operation. This file can be a compiler, a linker, or the name of a `.cmd` file you have written.

build server. The combination of a build processor and a build agent. See also *build agent* and *build processor*.

build target. The name of the part at the top of the build tree that is the final output of a build. TeamConnection uses the build target to determine the scope of the build. See also *build tree*.

build tree. A graphical representation of the dependencies that the parts in an application have on one another. If you change the relationship of one part to another, the build tree changes accordingly.

C

change control process. The process of limiting and auditing changes to parts through the mechanism of checking parts in and out of a central, controlled, storage location. Change control for individual releases can be integrated with problem tracking by specifying a process for the release that includes the tracking subprocess.

check-in. The return of a TeamConnection part to version control.

check-out. The retrieval of a version of a part under TeamConnection control. In nonconcurrent releases, the check-out operation does not allow a second user to check out a part until the first user has checked it back in.

child component. Any component in a TeamConnection family, except the root component, that is created in reference to an existing component. The existing component is the parent component, and the new component is the child component. A parent component can have more than one child component, and a child component can have more than one parent component. See also *component* and *parent component*.

child part. Any part in a build tree that has a parent defined. A child part can be input, output, or dependent. See also *part* and *parent part*.

client. A functional unit that receives shared services from a server. Contrast with *server*.

collision record. A status record associated with a work area or driver, a part, and one of the following:

- The work area or driver′s release
- Another work area

TeamConnection generates a collision record when a changed version of a part conflicts with a previously committed and integrated version of the same part. This is only related to *concurrent* development mode, when more than one developer can check out the same version of the same part concurrently. (In serial development, the part will be locked after the first check-out.)

command. A request to perform an operation or run a program from the command line interface. In TeamConnection, a command consists of the command name, one action flag, and zero or more attribute flags.

command line. (1) An area on the Tasks window or in the TeamConnection Commands window where a user can type TeamConnection commands. (2) An area on an OS/2 window where you can type TeamConnection commands.

committed version. The revision of a part that is visible from the release.

common part. A part that is shared by two or more releases, and the same version of the part is the current version for those releases.

comparison operator. An operator used in comparison expressions. Comparison operators used in TeamConnection are > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), and = (equal to).

component. A TeamConnection object that organizes project data into structured groups and controls configuration management properties. Component owners can control access to data and notification of TeamConnection actions. Components exist in a parent-child hierarchy with descendant components inheriting access and notification information from ancestor components. See also *access list* and *notification list*.

concurrent development. Several users can work on the same part at the same time. TeamConnection requires these users to reconcile their changes when they commit or integrate their work areas and drivers with the release. Contrast with *serial development*. See also *work area*.

configuration management. The process of identifying, managing, and controlling software modules as they change over time.

connect. The process of linking parts so that they are included in a build.

context. The current work area or driver used for part operations.

corequisite work areas. Two or more work areas designated as corequisites by a user so that all work areas in the corequisite group must be included as members in the same driver, before that driver can be committed. If the driver process is not used in the release, corequisite work areas must be integrated by the same command. See also *prerequisite work areas*.

current version. The last visible modification of a part in a driver, release, or work area.

current working directory. (1) The directory that is the starting point for relative path names. (2) The directory in which you are working.

D

daemon. A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their task; others operate periodically.

database. A collection of data that can be accessed and operated by a data processing system for a specific purpose.

default. A value that is used when an alternative is not specified by the user.

default query. A database search, defined for a specific TeamConnection window, that is issued each time that TeamConnection window is opened. See also *search*.

defect. A TeamConnection object used to formally report a problem. The user who opens a defect is the defect originator.

delete. If you delete a development object, such as a part or a user ID, any reference to that object is removed from TeamConnection. Certain objects can be deleted only if certain criteria are met. Most objects that are deleted can be re-created.

delta part tree. A directory structure representing only the parts that were changed in a specified place.

dependencies. In TeamConnection builds there are two types of dependencies:

- **automatic**. These are build dependencies that a parser identifies.
- **manual**. These are build dependencies that a user explicitly identifies in a build tree.

See also *build dependent*.

destroy. To remove the part record from the database on the TeamConnection server. The only TeamConnection development object that can be destroyed is a part.

disconnect. The process of unlinking parts so that they are not included in a build.

driver. A collection of work areas that represent a set of changed parts within a release. Drivers are only associated with releases whose processes include the track and driver subprocesses.

driver member. A work area that is added to a driver.

E

environment. (1) A user-defined testing domain for a particular release. (2) A defect field representing the environment where the problem occurred. (3) The string that matches a build agent with a build event.

environment list. A TeamConnection object used to specify environments in which a release should be tested. A list of environment-user ID pairs attached to a release, representing the user responsible for testing each environment. Only one tester can be identified for an environment.

explicit authority. The ability to perform an action against a TeamConnection object because you have been granted the authority to perform that action. Contrast with *base authority* and *implicit authority*.

extract. A TeamConnection action you can perform on a part, driver, or release. A part extraction results in copying the specified part to a client workstation. A driver extraction and release extraction result in all parts for the driver or release being copied to a designated location.

F

family. A logical organization of related data. A single TeamConnection server can support multiple families. The data in one family cannot be accessed from another family.

family administrator. A user responsible for all non-system-related tasks for one or more TeamConnection families, such as planning, configuring, and maintaining the TeamConnection environment and managing user access to those families.

family server. A workstation running the TeamConnection server software.

feature. A TeamConnection object used to formally request and record information about a functional addition or enhancement. The user who opens a feature is the feature originator.

file allocation table (FAT). The DOS- and OS/2-compatible file system that manages input, output, and storage of files on your system. File names can be up to eight characters long, followed by a file extension that can be up to three characters.

fix record. A status record associated with a work area and that is used to monitor the phases of change within each component affected by a defect or feature for a specific release.

freeze. The freeze command saves changed parts to the work area. Thus, TeamConnection takes a snapshot of the work area, including all of the current versions of parts visible from that work area, and saves this image of the system. The user can always come back to this stage of development in the work area. Note, however, that a freeze action does not make the changes visible to other users working in the release.

full part tree. A directory structure representing a complete set of active parts associated with the release.

G

graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real-world scene, often as a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

H

high-performance file system (HPFS). In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the existence of multiple, active file systems on a single personal computer, with the capacity of multiple and different storage devices. File names used with HPFS can have as many as 254 characters.

host. A host node, host computer, or host system.

host list. A list associated with each TeamConnection user ID that indicates the client machine that can access the family server and act on behalf of the user. The family server uses the list to authenticate the identity of a client machine when the family server receives a command. Each entry consists of a login, a host name, and a TeamConnection user ID.

host name. The identifier associated with the host computer.

I

implicit authority. The ability to perform an action on a TeamConnection object without being granted explicit authority. This authority is automatically granted through inheritance or object ownership. Contrast with *base authority* and *explicit authority*

import. To bring in data. In TeamConnection, to bring selected items into a field from a matching TeamConnection object window.

inheritance. The passing of configuration management properties from parent to child component. The configuration management properties that are inherited are access and notification. Inheritance within each TeamConnection family or component hierarchy is cumulative.

integrated problem tracking. The process of integrating problem tracking with change control to track all reported defects, proposed features, and subsequent changes to parts. See also *change control process*.

interest group. The list of actions that trigger notification to the user IDs associated with those actions listed in the notification list.

J

job queue. A queue of build scopes. One job queue exists for each TeamConnection family.

L

lock. An action that prevents editing access to a part stored in the TeamConnection development environment so that only one user can change a part at a time.

login. The name that identifies a user on a multiuser system. In OS/2, the login value is obtained from the TC_USER environment variable.

M

metadata. In databases, data that describe data objects.

N

name server. In TCP/IP, a server program that supplies name-to-address translation by mapping domain names to Internet addresses.

notification list. An object that enables component owners to configure notification. A list attached to a component that pairs a list of user IDs and a list of interest groups. It designates the users and the corresponding notification interest that they are being granted for all objects managed by this component or any of its descendants.

notification server. A server that sends notification messages to the client.

O

operator. A symbol that represents an operation to be done. See also *comparison operator*.

originator. The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a verification record. This responsibility can be reassigned.

owner. The user responsible for a TeamConnection object within a TeamConnection family, either because the user created the object or was assigned ownership of the object.

P

parent component. All components in each TeamConnection family, except the root component, are created in reference to an existing component. The existing component is the parent component. See also *child component* and *component*.

parent part. Any part in a build tree that has a child defined. See also *part* and *child part*.

parser. A tool that can read a source file and report back a list of dependencies of that source file. It frees a developer from knowing the dependencies one part has on other parts to ensure that a complete build is performed.

part. A collection of data that is stored by the family server and retrieved by a path name. Parts can be text objects, binary objects, and modeled objects. These parts can be stored by the user or the tool, or they can be generated from other parts, such as when a linker generates an executable file.

path name. The name of the part under TeamConnection control. A path name can be a directory structure and a base name or just a base name. It must be unique within each release.

prerequisite work areas. If a part is changed to resolve more than one defect or feature, the work area referenced by the first change is a prerequisite of the work area referenced by later changes. A work area is a prerequisite to another work area if:

- Part changes are checked in, but not committed, for the first work area.
- One or more of the same parts are checked out, changed, and checked in again for the second work area.

problem tracking. The process of tracking all reported defects through to resolution and all proposed features through to implementation.

process. A combination of TeamConnection subprocesses, configured by the family administrator, that controls the general movement of TeamConnection objects (defects, features, work areas, and drivers) from state to state within a component or release. See also *subprocess* and *state*.

Q

query. A request for information from a database, for example, a search for all defects that are in the open state. See also *default query* and *search*.

R

raw format. Information retrieved on the Report command that has the vertical bar delimiter separating field information, and each line of output corresponds to one database record.

refresh. This TeamConnection command updates a work area with any changes from the release. It also freezes the work area, if it is not already frozen.

relative path name. The name of a directory or a part expressed as a sequence of directories followed by a part name, beginning from the current directory.

release. A TeamConnection object defined by a user that contains all parts that must be built, tested, and distributed as a single entity.

restricted authority. The limitation on a user′s ability to perform certain actions at a specific component. Authority can be restricted by the superuser, the component owner, or a user with AccessRestrict authority. See also *authority*.

root component. The initial component created when a TeamConnection family is configured. All components in a TeamConnection family are descendants of the root component. Only the root component has no parent component. See also *component*, *child component*, and *parent component*

S

search. To scan one or more data elements of a set in a database to find elements that have certain properties.

serial development. While a user has parts checked out from a work area, no one else on the team can check out the part. The user develops new material without interacting with other developers on the project. TeamConnection provides the opportunity to hold the part until the user is sure that it integrates with the rest of the application. Thus, the lock is not released until the work area as a whole is committed. Contrast with concurrent development. See also *work area*.

server. A workstation that performs a service for another workstation.

shell script. A series of commands combined in a file that carry out a function when the file is run.

sizing record. A status record created for each component-release pair affected by a proposed defect or feature. The sizing record owner must indicate whether the defect or feature affects the specified component-release pair and the approximate amount of work needed to resolve the defect or implement the feature within the specified component-release pair.

stanza format. Data output generated by the Report command in which each database record is a stanza. Each stanza line consists of a field and its corresponding values.

state. Work areas, drivers, features, and defects move through various states during their life cycles. The state of an object determines the actions that can be performed on it. See also *process* and *subprocess*.

subprocess. TeamConnection subprocesses govern the state changes for TeamConnection objects. The design, size, review (DSR) and verify subprocesses are configured for component processes. The track, approve, fix, driver, and test subprocesses are configured for release processes. See also *process* and *state*.

superuser. This privilege lets a user perform any action available in the TeamConnection family.

system administrator. A user who is responsible for all system-related tasks involving the TeamConnection server, such as installing, maintaining, and backing up the TeamConnection server and the database it uses.

T

task list. The list of tasks displayed in the Tasks window. The user can customize this list to issue requests for information from the server. Tasks can be added, modified, or deleted from the lists.

TeamConnection client. A workstation that connects to the TeamConnection server by a TCP/IP connection and that is running the TeamConnection client software.

TeamConnection part. A part that is stored by the TeamConnection server and retrieved by a path name, release, type, and work area. See also *part*, *common part*, and *type*.

TeamConnection superuser. See *superuser*.

tester. A user responsible for testing the resolution of a defect or the implementation of a feature for a specific driver of a release and recording the results on a test record.

test record. A status record used to record the outcome of an environment test performed for a resolved defect or an implemented feature in a specific driver of a release.

track subprocess. An attribute of a TeamConnection release process that specifies that the change control process for that release will be integrated with the problem tracking process.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

type. All parts that are created through the TeamConnection GUI or on the command line will show up in reports with the type of *file* as the part type. The TeamConnection GUI and command line can only check in, check out, and extract parts of the *type* file.

> **Note:** Parts created through an API can have other specified types.

U

user exit. A user exit allows TeamConnection to call a user-defined program during the processing of TeamConnection transactions. User exits provide a means by which users can specify additional actions that should be performed before completing or proceeding with a TeamConnection action.

user ID. The identifier assigned by the system administrator to each TeamConnection user.

V

verification record. A status record that the originator of a defect or a feature must mark before the defect or feature can move to the closed state. Originators use verification records to verify the resolution or implementation of the defect or feature they opened.

version. (1) A specific view of a driver, release, or work area. (2) A revision of a part.

version control. The storage of multiple versions of a single part and information about each version.

view. An alternative and temporary representation of data from one or more tables.

W

work area. An object in TeamConnection that you create and associate with a release. When the work area is created, you see the most current view of the release and all the parts that it contains. You can check out the parts in the work area, make modifications, and check them back into the work area. You can also test the modifications without integrating them. Other users are not aware of the changes that you make in the work area until you integrate the work area to the release. While you work on files in a work area, you do not see subsequent part changes in the release until you integrate or refresh your work area.

working part. The checked-out version of a TeamConnection part.

Index

Printed in U.S.A.

SG24-4610-00