Joseph Morales
Documentation Sample

Following are sample pages from a programming language guide called *WFL Made Simple*. It was written as a supplement to a more formal and complete reference manual. At the time, Unisys Corporation was attempting to improve its user satisfaction ratings for software documentation, and the *Work Flow Language (WFL) Reference Manual* had been identified as a focus of customer complaints. The problems probably stemmed from a couple of causes:

- WFL was intended for a less technical audience than other programming languages, including operators and others with minimal programming experience.
- Programming reference manuals at Unisys were produced based on design documents created by the compiler team, who were chiefly interested in a formal and rigorous language description that incorporated complicated syntax diagrams.

Since I had previously done updates to the *WFL Reference Manual* and developed examples for it, I was in a good position to reformulate my understanding of the language in a more user-friendly format. We decided to take the *For Dummies* series of books as a model, and went with an approach that was

- Task and example based, using boldface type to indicate portions of examples that are called out in the descriptions that follow.
- Labeled with icons to indicate common information types, such as Tip, Caution, and Key idea.

Although this was a unique project for Unisys documentation at the time, I was able to work within existing corporate standards for format, typography, and style.

WFL Made Simple remains one of my favorite projects because I was able to break new ground and improve on previous approaches in our department.

Section 4
Running Tasks

This section explains the following topics related to initiating and controlling tasks:

- What is a task?
- Using the RUN statement
- Controlling file usage
- Using task equations
- Passing parameters to a task
- Using task variables
- Handling task terminations
- Running tasks in parallel

What Is a Task?

When you run a program from WFL, the system creates something called a *task*. A task is a single copy of a program that is executing in system memory. If you run the same program several times, you create a new task each time.

For a more detailed explanation of tasking concepts, refer to the *Task Management Programming Guide*

Using the RUN Statement

```
BEGIN JOB;
  RUN (WESLEY)OBJECT/ALTCOM ON SERV;
END JOB
```

Runs the program (WESLEY)OBJECT/ALTCOM ON SERV.

Key points to remember about the RUN statement:

- The RUN statement can initiate programs written in almost any programming language (ALGOL, C, COBOL, and so on).
- However, the RUN statement cannot initiate another WFL job; you must use the START statement for that purpose. Refer to "Starting a Job from Another WFL Job" in Section 2.
- The RUN statement must specify the name of the *object code file* for a program. The object code file is the compiled version of the program. By contrast, the *source code file* stores the program as it was originally written by the programmer. Object code files often have titles beginning with OBJECT or SYSTEM. (For information about compiling programs, refer to Section 8.)

Preventing Accidental Task Restarts

```
BEGIN JOB;
  ON RESTART,
     ABORT "Aborting job because of automatic restart";
  RUN OBJECT/PAYROLL;
END JOB
```

Runs the program OBJECT/PAYROLL no more than once. If the system restarts the job after a halt/load, the ON RESTART statement invokes an ABORT statement to terminate the job.

Rules

The ON RESTART statement is related to the automatic restart feature of WFL jobs. The system automatically restarts any WFL job that is interrupted by a halt/load. This automatic restart feature is helpful, provided that your job and its tasks are designed with possible restarts in mind. For information about how to design a job for restarts, refer to Section 15, "Designing Jobs for Restarts."

At this point, all you need to know is that automatic restarts can occur if you do not take steps to prevent them. For example, suppose that your WFL job runs a task that prints paychecks. If a halt/load occurs while the task is running, then after the halt/load, the job is restarted and runs the task again. The result could be that some paychecks are printed twice.

If your WFL job runs tasks that should not be restarted, you can use an ON RESTART statement such as the one in the preceding example. The ON RESTART statement specifies actions to be taken if the job is restarted after a halt/load. The ON RESTART statement is never executed unless a halt/load occurs.

Controlling File Usage

When you run a program from WFL, you can control what files the program uses and change the way that program uses files. You can

- Use file equations to specify the attributes of files used by the program.
- Use data specifications to supply input data for a program.
- Use task attributes to ensure that the program can successfully open a remote file.

Using File Equations

```
RUN (WESLEY)OBJECT/ALTCOM ON SERV;
   FILE INPUT(KIND=DISK,TITLE=(CHEN)COMDATA/INPUT);
   FILE RELCON(TITLE=(CHEN)COMDATA/RELCON ON LBFAM);
```

Runs a program, and uses file equations for two files used by the program.

`FILE`

A keyword indicating the start of a file equation.

`INPUT and RELCON`

Internal names of files used by the program. These names vary from one program to another.

Tip

To find out the internal names of files for a program, you must examine documentation for the program, talk to the person who created the program, or examine the source file for the program.

The internal name is usually the same as the identifier specified in the file declaration. For example, suppose a file is declared in the source file as FILE ABFAB (KIND=DISK, TITLE="DATA/INPUT/CFORM ON UFAM"). In this example, the internal name is ABFAB.

If the INTNAME attribute is specified, it overrides this default and specifies a different internal name. For example, the declaration FILE ECOM(KIND=DISK, INTNAME = "LATEREPORT") specifies a file with the internal name LATEREPORT.

`KIND`

A file attribute that specifies the type of medium on which the file resides. Possible values include DISK, TAPE, CD, PRINTER, and REMOTE, among

Running Several Tasks at the Same Time

If your WFL job runs multiple tasks, you have a choice between running the tasks one after another or running them at the same time (in parallel).

Flow of Control for Tasks

Figure 4–1 shows the flow of control for a job that runs tasks one after another. The job uses a RUN statement to initiate each task. The job itself is suspended and can do no work while each task runs.

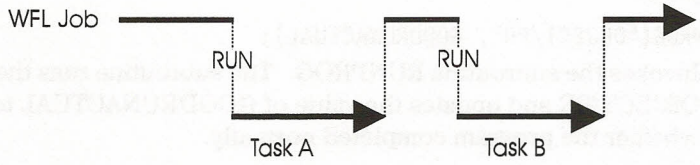


Figure 4–1. Running Tasks Sequentially

Figure 4–2 shows the flow of control for a job that runs two tasks at the same time. The job uses a PROCESS RUN statement to initiate each task. The job and the tasks run in parallel; that is, all can do work at the same time.

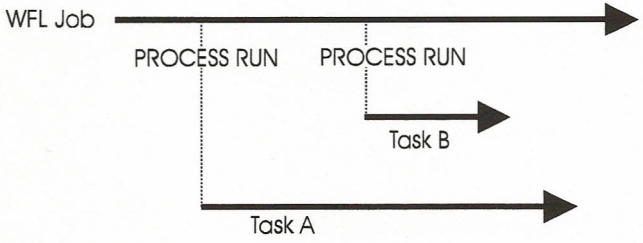


Figure 4–2. Running Tasks in Parallel

Tasks that run in parallel with the job are also known as *asynchronous tasks*.

CHANGE: Renaming Disk Files

The CHANGE statement changes the names of one or more files.

Renaming a Single File

```
CHANGE AUDIT/DB TO AUDIT/DAILY;
```

Changes the file with the name AUDIT/DB to the name AUDIT/DAILY.

Caution

If a file already exists with the new file name, the system deletes the existing file before making the change. The system does not request any confirmation before making the deletion.

This caution does not apply when you rename an entire directory. In that case, the system preserves existing files, as discussed in the following subsection.

Renaming a Directory

```
CHANGE (OPS)RESULTS/= TO (OPS)SAVE/=;
```

Changes all the files in the directory RESULTS/= to the directory SAVE/=.

Key Idea

MCP servers have no permanent directories of the type you find on Windows or UNIX systems. In this manual, the term *directory* is simply shorthand for a collection of files whose titles begin with one or more of the same elements. Thus, the CHANGE statement shown above renames files whose titles begin with (OPS)RESULTS. There is no need to separately rename the directory itself.

Rules

If the new directory name already exists, the files are added to that directory. Any files that already belonged to the new directory remain unchanged.

When you rename a directory of files, the system avoids overwriting any existing file. The system renames as many of the files as it can without creating conflicts, and leaves the other file names unchanged.

Example of Effects

The following table shows the effects of this statement on a set of files. The Before column lists the names of the files before the CHANGE statement is executed. The After column lists the names of the files after the CHANGE statement is executed.

Before	After
RESULTS/INPUT	RESULTS/INPUT (name is unchanged because of conflict with existing file SAVE/INPUT)
RESULTS/USERS	SAVE/USERS
RESULTS/DEVCON	SAVE/DEVCON
SAVE/INPUT	SAVE/INPUT
SAVE/QDATA	SAVE/QDATA

Renaming Files on a Specified Family

```
CHANGE WEEKLY/REPORT ON USERC TO DAILY/REPORT;
```

Changes the name of a file on family USERC.

Rules

The *ON <family>* part, if it is included, must follow the old file name rather than the new file name.

If an *ON <family>* part is not included, the file is assumed to be on family DISK.

Family Substitution for CHANGE Statements

```
BEGIN JOB;
  FAMILY DISK = ACCT OTHERWISE DEVPK;
  CHANGE DAILY/REPORT TO DAILY/BACK;
END JOB
```

Changes the name of DAILY/REPORT ON ACCT to DAILY/BACK ON ACCT. The system does not search DEVPK, even if the file is not present on ACCT.

Copy Request Basics

The following pages explain how to

- Submit copy requests from CANDE, MARC, and the ODT.
- Copy single files, multiple files, or directories of files
- Copy all files from your own usercode or from another usercode
- Copy all files on a family or all usercoded files on a family.
- Rename files as they are copied.
- Copy files between multiple sources and destinations.
- Combine separate copy requests.

Submitting Copy Requests

You can submit the COPY statement as part of a WFL job or as a command in CANDE, MARC, NX/TaskCenter, or at an ODT. For information about entering COPY as a command, refer to "Sources for Submitting WFL Statements or Jobs" in Section 2, "Submitting WFL Statements and Jobs."

In general, you should store the COPY statement in a formal WFL job if the statement is complex or if you plan on using it repeatedly. Otherwise, it is easier to simply type in the COPY statement as a command.

Copying Single Files

```
COPY OBJECT/PAYROLL/REPORT FROM DBFAM(PACK) TO SERV(PACK);
```

Creates a copy of OBJECT/PAYROLL/REPORT on SERV family. The source file remains present on DBFAM.

Note

If a file with the requested name already exists at the destination, the COPY statement overwrites that file. If you want to prevent this overwriting, use the ADD statement as described under "Preserving Existing Files at the Destination" later in this section.

Copying Multiple Files

```
COPY OBJECT/PAYROLL/REPORT, DATA/INPUT FROM DBFAM(PACK) TO
SERV(PACK);
```

Copies the files OBJECT/PAYROLL/REPORT and DATA/INPUT from one disk family to another.

Copying Directories

```
COPY DATA/= FROM DBFAM(PACK) TO SERV(PACK);
```

Copies all the files in the directory DATA/= from one disk family to another. Does not copy the file DATA itself (if there is one).

Copying All Files from Your Own Usercode

```
COPY = FROM DBFAM(PACK) TO SERV(PACK);
```

Copies all the files under the usercode of the WFL job from DBFAM family to SERV family. WFL jobs typically run under the usercode of the MARC or CANDE session where you started the job.

Note

If you start a job from an ODT, the WFL job typically runs without a usercode. For such a job, *COPY* = copies all files on the family, including both usercoded and nonusercoded files.

Copying All Files from Another Usercode

```
COPY (SMITHJA)= FROM DBFAM(PACK) TO SERV(PACK);
```

Copies all the files under the usercode SMITHJA from DBFAM family to SERV family.

Rule

If the usercode being copied from is different from the usercode of the job, then the job must have special privileges. Refer to "COPY or ADD" in Section 13, "Security."

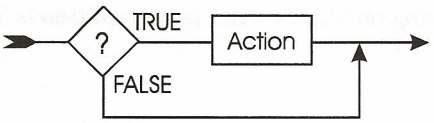
IF: Making a Choice

Using IF statements, you can

- Make a choice, based on whether a particular Boolean value is TRUE or FALSE.
- Provide an alternative if the value is FALSE.
- Test several Boolean values by stringing IF statements together or nesting them.

The Simple IF Statement

Logic Flow



Executes an action if a test condition is TRUE. Otherwise, skips past that action.

Example

```
RUN OBJECT/VALIDATE [T];
IF T(TASKVALUE = 1) THEN
  BEGIN
    PRINT VALID/DATA/=;
    REMOVE TEMP/VALIDATE;
  END;
```

Executes a PRINT statement and a REMOVE statement if the task OBJECT/VALIDATE set its TASKVALUE task attribute to 1.

Rules

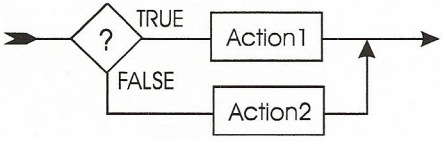
The IF statement executes another statement if a particular condition is true. You specify the condition that should be checked, and the statement that should be executed if the condition is true.

Test Condition

For descriptions of the various types of conditions that can be tested in an IF statement, refer to "Booleans: Evaluating Truth Conditions" in Section 11, "Working With Data."

Providing an Alternative with IF ELSE

Logic Flow



Executes one of two actions, depending on whether the test condition is TRUE or FALSE.

Example

```
IF FILE TRANS/STATUS IS RESIDENT THEN
   RUN OBJECT/TRANS;
ELSE
   ABORT "NO FILE TRANS/STATUS";
```

Checks whether a file is resident, and takes one of two different actions, depending on the result.

WAIT: Interrupting the Job Temporarily

Logic Flow

Stops execution of the job until a particular condition is fulfilled.

Waiting for a File

```
WAIT (FILE TEMP/INDEX/HTML ON SERV IS RESIDENT);
```

Waits until the file TEMP/INDEX/HTML is present on SERV family.

Waiting for an Operator OK

```
WAIT (OK);
```

Waits for the operator to enter a *<mix number> OK* command.

Waiting for a Length of Time

```
WAIT (120);
```

Waits for 120 seconds (that is, two minutes).

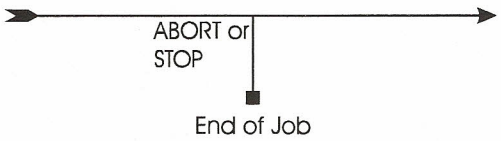
Waiting for Tasks

The WAIT statement has several features that allow the job to wait for parallel tasks. Refer to the following topics in Section 4, "Running Tasks":

- "Waiting for Tasks to Terminate"
- "Waiting for Changes in Task Status"
- "Waiting for a Task Attribute Value"

ABORT and STOP: Interrupting the Job Permanently

Logic Flow



Ends execution of the job. Any remaining statements in the job are never executed.

Example

```
RUN OBJECT/PREPARE [T1];
IF T1 IS NOT COMPLETEDOK THEN
   ABORT "Job aborted due to failure of OBJECT/PREPARE";
RUN OBJECT/SORTDATA [T2];
IF T2 IS NOT COMPLETEDOK THEN
   STOP "Job terminated due to failure of OBJECT/SORTDATA";
RUN OBJECT/REPORT;
```

The ABORT and STOP statements differ only in the termination messages they produce.

- An ABORT statement causes the system to display a P-DS termination message, which is typical of an abnormal termination.
- A STOP statement causes the system to display an EOJ termination message, which is typical of a normal termination.