The Mainframe Software Partner
For The Next 50 Years

ISPW
Interfaces Guide

Release 17.02

Please direct questions about ISPW
or comments on this document to:

Compuware Customer Support

https://go.compuware.com/

Doc. OCT2018
September 26, 2018

Contents

Introduction

This manual documents the various interfaces available in ISPW, and contains the following chapters:

- Chapter 1, “External Call Interface”
- Chapter 2, “Standalone Load Modules”
- Chapter 3, “DB2 Programs, Plans, and Packages”
- Chapter 4, “DB2 Generated DECLARE Statements”
- Chapter 5, “Telon Interface”
- Chapter 6, “QMF Interface”
- Chapter 7, “Natural Interface”
- Chapter 8, “ISPW Web Interface”.

Related Publications

- *ISPW Release Notes:* Overview of release features, supported operating environments, customer support, and any new ISPW information.
- *ISPW Installation and Configuration Guide*: Gives step-by-step instructions for the system programmer to configure, customize, and maintain ISPW. Refer to it when installing ISPW according to the *Compuware Installer Mainframe Products SMP/E Installation Guide*.
- *ISPW Deploy Reference*: Introduces ISPW users to the new ISPW Deploy product. It gives details of the concepts and facilities from both an end-user and technical perspective.
- *ISPW Messages and Codes*: Documents the messages and codes for ISPW, including started task errors, abend codes, return codes, allocation errors, and ISPW CM errors.
- *ISPW Planning Guide*: Provides information for use in the early stages of an ISPW implementation. It contains only what is necessary for planning and initial installation.
- *ISPW Remote Server Guide*: Describes ISPW remote servers, Controlled Tasks run on platforms separate from the main ISPW Administration Server.
- *ISPW Technical Reference*: Provides detailed information on ISPW’s structure, terms and concepts, maintenance functions, processing, security, and other technical content.
- *ISPW User Guide*: Provides an overview for Applications and other Information Systems staff to use ISPW effectively.
- *ISPW Upgrade Guide, Release 4.4 to 17.02*: Describes the major differences between ISPW 4.4 and ISPW 17.02 and serves as a guide for the upgrade process between these versions.
- *ISPW Web Interface Installation and Configuration Guide*: Provides instructions on how to install the ISPW Web Interface. ISPW Web is an Internet-based application designed to be used on workstations or smartphones. The ISPW Web Interface uses a Web browser that enables you to remotely approve or reject ISPW actions as well as deploy software to both mainframe and distributed environments.

Online Documentation

The ISPW product installation package does not include the product documentation. Access the ISPW documentation from the Compuware FrontLine Customer Solutions website at **https://go.compuware.com** in the following electronic formats:

- Release Notes in HTML format
- Product manuals in PDF format
- Adobe PDF index file (PDX file).

The product documentation is available for viewing or downloading:

- View PDF files with the free Adobe Reader, available at **http://www.adobe.com**.
- View HTML files with any standard web browser.

Customer Support

Compuware provides a variety of support resources to make it easy for you to find the information you need.

Compuware Support Center Website

You can access online information for Compuware products via the Compuware Support Center website at **https://go.compuware.com**.

Compuware Support Center provides access to critical information about your Compuware products. You can review frequently asked questions, read or download documentation, access product fixes, or e-mail your questions or comments. The first time you access Compuware Support Center, you are required to register and obtain a password. Registration is free.

Contacting Customer Solutions

Phone

- USA and Canada: 1-800-538-7822 or 1-313-227-5444.
- All other countries: Contact your local Compuware office. Contact information is available at **https://go.compuware.com**.

Web

You can report issues via the Quick Link **Create & View Support Cases** on the Compuware Support Center home page.

Note: Please report all high-priority issues by telephone.

Mail

Compuware Customer Solutions
Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

Corporate Website

To access Compuware's site on the Web, go to **http://www.compuware.com**.

The Compuware site provides a variety of product and support information.

Chapter 1.
External Call Interface

The ISPW External Call Interface (ECI) is an alternate interface to the User Interface that can be used to perform a subset of ISPW functions. The ECI can be called either by an API (programmatically) or via JCL. The ECI is an MVS Interface to ISPW.

When to Use the ECI

The ECI is intended for those situations in which processing occurs outside of ISPW that needs to be reflected in the ISPW repository. This processing might be a result of external events or could be a set of mass operations which have been done outside the user interface.

Example 1

When a component has been implemented into another environment, the status of the Task can be updated to reflect this by an ECI call.

Example 2

When migrating application systems into ISPW, the ECI can be used to populate the repository with information about the components.

Limitations

The ECI only affects ISPW's logical meta-data repository. For functions concerning components, it does not provide any consistency checking with the actual component itself. It is the user's responsibility to ensure that the information provided in the ECI calls is correct.

Using the ECI via the API

Calling the ECI

Calls to ISPW functions are provided though the WZZECI interface module.

This is an LE-compliant routine that can be called (either statically or dynamically) from any other LE program.

All calls to WZZECI must include a common data structure in which an OBJECT and METHOD identify the type of call. Other parameters may be required, depending upon the type of call.

All ECI sessions must begin with a call to initialize the ISPW environment and end with a call to clean up the session. Between these two calls, any number of other calls may be made.

Processing Considerations

After successful initialization of the ISPW environment, multiple ECI functions can be performed. Each ECI function is considered to be separate and, if successful, is committed in the repository.

Security

All ECI calls pass through authorization checking by the ISPW Server. It is recommended that security rules and associated SAF profiles be defined to protect usage of the ECI. Authority checking is done against the UserID making the ECI calls.

Handling Errors

Calls made to the ECI which are not successful result in non-zero return codes. Error messages are returned to help determine what the error is.

Using the ECI via JCL

ECI calls via JCL are made though the WZZECIJ program.

Figure 1-1 shows an example of the ASGNMENT ADD function.

Figure 1-1. Task Operation - Initiate and Task Operation - Specify Tasks JCL for WZZECIJ

```
//WZZECIJ  JOB MSGCLASS=X,CLASS=A,NOTIFY=JohnDoe
//STEP1    EXEC  PGM=WZZRCJOB,PARM='Default/WZZECIJ'
//STEPLIB DD DSN=ISPW.CONFIG.AUTHLOAD,DISP=SHR
//*****************************************************************
//* STEP 1:  PERFORM TASK OP IN BATCH VIA ECI
//*****************************************************************
//WORKIN   DD *
$DEFINE_TSB
  APPLID=TST
  STRMNAME=TEST
  OP=X
  LVL=HLD1
$DEFINE_TSBT
  APPLID=TST
  MTYPE=COB
  MNAME=TESTCOB
```

The format of the input via JCL is the `$DEFINE_`*`datagroup`* followed by a line for each keyword and its corresponding value. Valid `$DEFINE_`*`datagroup`* values are specified alongside the descriptions of the functions on the following pages. Each `$DEFINE_`*`datagroup`* statement delimits any previous call. The indentation of the keywords is for clarity purposes only.

The valid keywords for each call are documented in the following descriptions of each function call. They can be specified in any order. Values that are text with embedded spaces must be enclosed by single quotes.

Limitations

The JCL Interface does not support all ECI calls. The description for each function specifies whether the JCL Interface is valid for that function. Some calls return values (for

example, Assignment Add returns details about the Assignment just added). The JCL Interface cannot do anything with these values.

Connecting to the ISPW Environment

Initiation of WZZECI creates an ISPW Environment automatically using the connection information specified against the WZZSCD DDName. The format of the WZZSCD file is the same as used for TSO users.

Security

All ECI calls pass through authorization checking by the ISPW Server just like a normal ISPW user. Authority checking is done against the UserID of the batch Job.

Handling Errors

Calls made to the ECI which are not successful result in non-zero return codes. Error messages are displayed in the output to help determine what the error is.

ECI Functions and Calls

Functions

Usage of the ECI is described in terms of ***functions***. Each function is considered to be a "unit of work". Some functions relate to a single ECI call, while others may consist of multiple calls.

Calls

ECI Calls are distinguished by the Object and Method specified with the call.

Function and Call Mappings

Table 1-1 summarizes the Functions currently supported with their associated Objects and Methods. The **JCL Call** column indicates whether the function can be called via the JCL Interface program WZZECIJ.

Table 1-1. Supported Functions

Function	Object	Method	Description	JCL Call
Initialize Environment	ENV	INIT	Initialize the ISPW environment	No
Terminate Environment	ENV	TERM	Cleanup the ISPW environment	No
Assignment Add	ASGNMENT	ADD	Add and Assignment	Yes
Release Add	RELEASE	ADD	Add a Release	Yes
Task Load	TASK	LOAD	First call in loading of a single Task	Yes
	TASK	LOADPART	Calls to load individual parts of a Component for a Task (for example, load, DBRM, etc.)	Yes
	TASK	LOADEND	Delimit the loading of a single Task	Yes
Task Status Update	TASK	STATUPDT	Update the status of a Task (for example, the result of a generate)	No

Table 1-1. Supported Functions (Continued)

Function	Object	Method	Description	JCL Call
Task Operation	TASK	BATOP	First call in processing Task operations in a Set	Yes
	TASK	BATOPTSK	Call for each Task to have operation performed	Yes
	TASK	BATOPEND	Delimit the list of Tasks and Lock the set for execution.	Yes
Set Approve	SET	APRVMOD	Approve/Deny Set	Yes
Set Execute	SET	EXECUTE	Release Set for Execution	Yes
Component Transport Control	TRNSPCTL	AUTHREQ	Manage the operation of Component Transport Servers.	No

Initialize Environment

Purpose

This call initializes an ISPW environment. It must be successfully made before any of the other ECI functions can be called. The JCL Interface makes this call implicitly, which is why an explicit JCL call of this type is not available.

API Parameter List

ECIR

The ECIR Data Structure must have the values in Table 1-2 set.

Table 1-2. ECIR Data Structure Values for Initialize

Parameter	Value
CONNECTION DEFINITION	DDNAME of the ISPW Connection Definition file
OBJECT	ENV
METHOD	INIT

ISPW Connection Definition

The connection definition is a sequential file or PDS member containing the necessary information to connect to an ISPW server. This dataset must be allocated to the DDNAME specified in the ECIR.

The specification is made by keyword parameters starting in column 1 in the following format:

```
CMID = cccc
SRID = ssss
```

where the parameters and defaults in Table 1-3 apply.

Table 1-3. Connection Definition Parameter Defaults

Parameter	Default	Description
cccc	ISPW	ISPW Communications MVS Subsystem ID
ssss	ISPW	ISPW Server ID

ECIR Data Structure

The same data area must be passed on each call as ISPW uses this as an anchor to maintain the ISPW environment. When a non-zero return code is returned from WZZECI, error information is passed back in the ECIR. The ECIR layout is described in Table 1-4.

Table 1-4. ECIR Layout

Position	Length	Type	Field	Description
1	4	CHAR	EYECATCHER	Used to identify the ECIR control block. Must have a value of **ECIR**.
5	4	BINARY	ENVIRONMENT HANDLE	Used by ISPW to maintain environment. The ENV INIT call populates this field and it is used by all subsequent ECI calls as a means of identifying the session. This should ***not*** be initialized between calls.
9	8	CHAR	CONNECTION DEFINITION	Specifies the ISPW Connection Definition Name. Must be set on ENV INIT request to DDNAME containing connection definition
17	8	CHAR	OBJECT	Specifies the request Object name
25	8	CHAR	METHOD	Specifies the request Method name
33	4	BINARY	RETURN CODE	ISPW request return code.
37	4	BINARY	REASON CODE	ISPW request reason code.
41	8	CHAR	ERROR ID	Error ID as defined in the Error Messages
49	8	CHAR	ERROR CODE	ISPW Diagnostic Code
57	72	CHAR	SHORT ERROR MSG	Short error message
129	255	CHAR	LONG ERROR MSG	Long error message
384	32	CHAR	RESERVED	

Return Codes

For each ECI request, the return codes in Table 1-5 are possible.

Table 1-5. ECI Request Return Codes

Return Code	Description
0	Successful
4	Invalid ECIR
8	ISPW Server Function error (See error message)
12	ECI Usage Error (See Reason Codes)
16	Error in Environment Initialization
20	Internal ISPW Communications Error

Possible Reason Codes

For non-zero Return codes, a Reason code (see Table 1-6) is supplied which provides more information on the cause of the error.

Table 1-6. ECI Request Reason Codes

Reason Code	Description
1	ECI Initialization not done
2	Bad OBJECT or METHOD
3	Bad Field Map (Internal Error)

Terminate Environment

Purpose

This call cleans up and terminates an ISPW environment. It should be made after all other calls for the session have been done. The JCL Interface makes this call implicitly, which is why an explicit JCL call of this type is not available.

API Parameter List

ECIR

The ECIR Data Structure must have the values in Table 1-7 set.

Table 1-7. ECIR Data Structure Values for Terminate

Parameter	Value
OBJECT	ENV
METHOD	TERM

Add Assignment

Purpose

This call adds an Assignment in ISPW.

API Parameter List

ECIR, ASA

The ECIR Data Structure must have the values in Table 1-8 set.

Table 1-8. ECIR Data Structure Values for Add Assignment

Parameter	Value
OBJECT	ASGNMENT
METHOD	ADD

JCL Invocation

```
$DEFINE_ASA
```

ASA Data Structure

The ASA data area contains details required to add the Assignment. The ASA layout is described in Table 1-9.

Table 1-9. ASA Layout

Position	Length	Type	Description	JCL Keywords	API	JCL
1	4	CHAR	Default Application for Tasks added to this Assignment	APPLID	Mandatory	Mandatory
5	50	CHAR	Description for Assignment	DESC	Optional	Optional
55	4	CHAR	Default signout level for Tasks added to this assignment	DFLTLVL	Optional	Optional
59	8	CHAR	Work Reference ID – text field for customer use	REFNO	Optional	Optional
67	8	CHAR	UserID of the Assignment Owner Defaults to UserID of caller	OWNER	Optional	Optional
75	10	CHAR	Default Release ID associated to Tasks added to this Assignment	IMPLNO	Optional	Optional
85	4	CHAR	Assignment Prefix – not required if a specific Assignment is specified	ASGNPRF	Optional*	Optional*
89	8	CHAR	Stream Name	STRMNAME	Mandatory	Mandatory
97	10	CHAR	Assignment Number – returned if Prefix is specified, otherwise must be specified	PROJNO	Optional*	Optional*
107	10	CHAR	Return Value – Creation Date		Optional	NA
117	8	CHAR	Return Value – Creation Time		Optional	NA

* You must specify either Assignment Number or Assignment Prefix.

Add Release

Purpose

This call adds a Release in ISPW.

API Parameter List

ECIR, RLA

The ECIR Data Structure must have the values in Table 1-10 set.

Table 1-10. ECIR Data Structure Values for Add Release

Parameter	Value
OBJECT	RELEASE
METHOD	ADD

JCL Invocation

```
$DEFINE_RLA
```

RLA Data Structure

The RLA data area contains details required to add the Release. The RLA layout is described in Table 1-11.

Table 1-11. RLA Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	10	CHAR	Release ID – returned if release prefix is used, otherwise must be specified.	RLSEID	Optional*	Optional*
11	50	CHAR	Description for Assignment	DESC	Optional	Optional
61	4	CHAR	Owning Application for Release	APPLID	Mandatory	Mandatory
65	8	CHAR	Work Reference ID – text field for customer use	REFNO	Optional	Optional
73	8	CHAR	UserID of the Assignment Owner Defaults to UserID of caller	OWNER	Optional	Optional
81	8	CHAR	Stream Name	STRMNAME	Mandatory	Mandatory
89	10	CHAR	Implementation Date – informational only	IMPLDATE	Optional	Optional
99	8	CHAR	Implementation Time – informational only	IMPLTIME	Optional	Optional
107	4	CHAR	Release Prefix – must be specified if Release ID is not	RLSEPRF	Optional	Optional

* You must specify either Release ID or Release Prefix.

Task Load

Purpose

The Task load process is used to populate the ISPW repository with Tasks and Components. Tasks can be loaded at any level (even production). It is the responsibility of the caller to make sure that Task information loaded reflects the actual location and status of the physical components.

Description

The Task load process consists of one TASK LOAD call, zero or more TASK LOADPART calls, and one TASK LOADEND call. The information for the Task is not committed to the repository until a successful TASK LOADEND call is made.

Version Sequence Issues

Loading Tasks through the ECI has consequences for versioning, in that these tasks are inherently not based upon any existing versions in the repository.

ECI loaded tasks are considered to be ***mutable*** (for example, ISPW cannot guarantee the integrity of the task with relation to other tasks), and so the strict versioning sequence rules do not apply.

Task Load – Initiate

Purpose

This call initiates the loading of a Task into ISPW. It can be made to add a Task and its associated Component information to any level, including Production.

API Parameter List

ECIR, TSI

The ECIR Data Structure must have the values in Table 1-12 set.

Table 1-12. ECIR Data Structure Values for Task Load – Initiate

Parameter	Value
OBJECT	TASK
METHOD	LOAD

JCL Invocation

```
$DEFINE_TSI
```

TSI Data Structure

The TSI data area contains details required to add the Task. The TSI layout is described in Table 1-13.

Table 1-13. TSI Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	4	CHAR	ISPW Application ID	APPLID	Mandatory	Mandatory
5	8	CHAR	Stream Name	STRMNAME	Mandatory	Mandatory
13	4	CHAR	ISPW Component Type (must be a valid Component Type for the specified Application)	MTYPE	Mandatory	Mandatory
17	4	CHAR	Reserved – set to null or spaces	MCLAS	NA	NA
21	64	CHAR	Component Name	MNAME	Mandatory	Mandatory
85	8	CHAR	Component Short Name (otherwise spaces). If this is spaces, ISPW will automatically generate the short name, which will be the same as the Component Name for Component Names that are 8 characters or less. For Component Names longer than 8 characters, ISPW will generate a unique name.	SHRTNAME	Optional	Optional
93	10	CHAR	Owning Assignment ID (must be a valid Assignment number)	PROJNO	Mandatory	Mandatory
103	10	CHAR	Associated Release ID (must be either a valid Release ID or spaces)	RLSEID	Optional	Optional
113	1	CHAR	Action Code. space – normal ISPW processing **D** – Component to be deleted **C** – Component source in production	ACTION	Optional	Optional
114	4	CHAR	Level the Component is at (must be a valid Level for the specified Application)	CLVL	Mandatory	Mandatory
118	4	CHAR	Level the Component logically started. Defines the "Path" the Component moves to production (must be a valid starting level for the specified Application)	SLVL	Mandatory	Mandatory

Table 1-13. TSI Layout (Continued)

Position	Length	Type	Description	JCL Keyword	API	JCL
122	4	CHAR	Target environment for generated objects	TENV	Optional	Optional
126	2	CHAR	ISPW Operation code. If spaces, ISPW will use **TL**.	LASTOP	Optional	Optional
128	10	CHAR	Date of last ISPW operation – YYYY-MM-DD. If spaces, ISPW will use the current date.	LASTDATE	Optional	Optional
138	8	CHAR	Time of last ISPW operation – HH:MM:SS. If spaces, ISPW will use the current time.	LASTTIME	Optional	Optional
146	8	CHAR	UserID identifying who did last operation. If spaces, ISPW will use the ECI caller UserID.	USERID	Optional	Optional
154	8	CHAR	User-specified external version number	EXTVER	Optional	Optional
162	50	CHAR	For new components in ISPW, this will become the component description. (For example, if doing a TASK LOAD for an already existing component, this field has no effect.)	CMPNDESC	Optional	Optional
212	50	CHAR	Text field describing reason for Task	CHNGDESC	Optional	Optional
262	8	CHAR	UserID identifying who did the checkout. If spaces, ISPW will use the ECI caller UserID.	COPYUSER	Optional	Optional
270	10	CHAR	Checkout Date – YYYY-MM-DD. If spaces, ISPW will use the current date.	COPYDATE	Optional	Optional
280	8	CHAR	Checkout Time – HH:MM:SS. If spaces, ISPW will use the current time.	COPYTIME	Optional	Optional
288	1	CHAR	Indicates whether the Task being loaded is to be marked as successfully generated or not: **C** – successfully generated **R** – a generate is required blank – not relevant	GENSTAT	Optional	Optional
289	1	CHAR	Indicates whether the Task being loaded is to be marked as successfully implemented or not: **C** – successfully implemented **R** – implementation is required blank – not relevant	IMPLSTAT	Optional	Optional
290	1	CHAR	Indicates whether any Tasks superseded by the one being loaded are to be cleaned up automatically: **Y** – cleanup superseded Task automatically **N** or space – will not clean up the superseded Task	CUIND	Optional	Optional
291	6	CHAR	Return value containing the internal ISPW Task ID	TASKID	Optional	NA
297	6	CHAR	Return value containing the internal ISPW Component ID	CMPNID	Optional	NA
303	6	CHAR	Must be set to null (x'00'). This input field will be exploited in a future ISPW release.	OCMPNID	Optional	NA

Table 1-13. TSI Layout (Continued)

Position	Length	Type	Description	JCL Keyword	API	JCL
309	64	CHAR	Must be spaces. This input field will be exploited in a future ISPW release.	CRELPATH	Optional	Optional

Task Load – Load Parts

Purpose

This (optional) call is only used in conjunction with a TASK LOAD. It is used to define the individual parts associated with a component (for example, DBRM, load module, listing, etc.). Each Part must be defined in accordance with the Associations defined in ISPW for the specific Component Type and Application the Part is associated with. Zero to many calls of this type follow a TASK LOAD.

API Parameter List

ECIR, TSIP

The ECIR Data Structure must have the values in Table 1-14 set.

Table 1-14. ECIR Data Structure Values for Task Load – Load Parts

Parameter	Value
OBJECT	TASK
METHOD	LOADPART

JCL Invocation

```
$DEFINE_TSIP
```

TSIP Data Structure

The TSIP data area contains details required to add the Parts associated with a Component. The TSIP layout is described in Table 1-15.

Table 1-15. TSIP Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	4	CHAR	Target Environment, as indicated in the ISPW Associations, to correctly identify where this Part is stored	TENV	Optional	Optional
5	4	CHAR	Set value as specified in the ISPW Associations which relates to this Part (for example, GMT1)	ASSCSET	Mandatory	Mandatory
9	1	CHAR	Sequence value as specified in the ISPW Associations which relates to this Part (for example, 2)	ASSCSEQ	Mandatory	Mandatory
10	64	CHAR	Name of the Part	PARTNAME	Mandatory	Mandatory

Task Load – Delimit the Loading for a Task

Purpose

This call is only used in conjunction with a TASK LOAD (and any associated TASK LOADPART calls). It is used to delimit the loading of a Task and contains the User Extension Data. Every TASK LOAD call must be completed by a TASK LOADEND call.

API Parameter List

ECIR, TSIE

The ECIR Data Structure must have the values in Table 1-16 set.

Table 1-16. ECIR Data Structure Values for Task Load – Delimit the Loading for a Task

Parameter	Value
OBJECT	TASK
METHOD	LOADEND

JCL Invocation

```
$DEFINE_TSIE
```

Only the $DEFINE_TSIE is required. No additional parameters are necessary, unless User Extension Data is to be updated for this module.

TSIE Data Structure

The TSIE data area contains user data associated with a Task and is used to delimit the sequence of calls for a single Task. The TSIE layout is described in Table 1-17.

Table 1-17. TSIE Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	254	CHAR	User extension data	VPS01U	Optional	Optional
255	254	CHAR	User extension data	VPS02U	Optional	Optional
509	254	CHAR	User extension data	VPS03U	Optional	Optional
763	254	CHAR	User extension data	VPS04U	Optional	Optional

Task Status Update

Purpose

The TASK STATUPDT call performs a Task status update.

API Parameter List

ECIR, TSUS

The ECIR Data Structure must have the values in Table 1-18 set.

Table 1-18. ECIR Data Structure Values for Task Status Update

Parameter	Value
OBJECT	TASK
METHOD	STATUPDT

TSUS Data Structure

The TSUS data area contains details required to perform the status update. The TSUS layout is described in Table 1-19.

Table 1-19. TSUS Layout

Position	Length	Type	Description
1	6	BINARY	ISPW TASK IDENTIFIER
7	8	BINARY	ISPW PROCESS TOKEN
15	2	CHAR	OPERATION CODE of operation in progress
17	1	CHAR	**Y** to specify that operation processing is completed
18	1	CHAR	**Y** to specify that operation completed successfully
19	1	CHAR	Usually **B** to indicate message from background process
20	16	CHAR	Text of message to be set on Task
36	25	CHAR	

Task Operation

Purpose

The Task Operation process is used to perform an ISPW operation against one or more ISPW Tasks. A Set is created, the Tasks are put into the Set, and the Set is locked for execution.

Description

The Task Operation process consists of one TASK BATOP call (to create the Set), one or more TASK BATOPTSK calls (to put Tasks into the Set), and one TASK BATOPEND call. Once the BATOPEND call is made, the Set is locked for execution.

Handling Error Situations

This set of calls creates a Set and then attempts to put each Task into it for the requested Operation. This can lead to various errors and warnings (for example, overlay messages). A flag on the TASK BATOP is used to determine what ISPW will do in these situations (for example, continue, stop, or completely roll back the creation of the Set).

Task Operation – Initiate

Purpose

This call initiates the Task Operation process. Its input contains fields that specify the type of operation, level at which the operation is to be performed, and the date/time that the operations will be executed. This corresponds to the Set definition.

API Parameter List

ECIR, TSB

The ECIR Data Structure must have the values in Table 1-20 set.

Table 1-20. ECIR Data Structure Values for Task Operation – Initiate

Parameter	Value
OBJECT	TASK
METHOD	BATOP

JCL Invocation

```
$DEFINE_TSB
```

TSB Data Structure

The TSB data area contains details about the Set that will be created. The TSB layout is described in Table 1-21.

Table 1-21. TSB Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	4	CHAR	Application ID	APPLID	Mandatory	Mandatory
5	8	CHAR	Stream Name	STRMNAME	Mandatory	Mandatory
13	2	CHAR	Operation (for example, **P** for Promote)	OP	Mandatory	Mandatory
15	4	CHAR	Level at which the operation is to be performed	LVL	Mandatory	Mandatory
19	8	CHAR	Set Owner. When using the ECI, this is always set to the userID of the ECI caller. Other values coded here are ignored.	OWNER	Optional	Optional
27	50	CHAR	Set Description. If not specified, will be set to **BATCH REQUEST FOR TASK OP** ***op***.	DESC	Optional	Optional
77	10	CHAR	Set Start Date. If not specified, will be set to current date.	STRTDATE	Optional	Optional
87	8	CHAR	Set Start Time. If not specified, will be set to current time.	STRTTIME	Optional	Optional
95	10	CHAR	Deploy Implementation start date	DIMPDATE	Optional	Optional
105	8	CHAR	Deploy Implementation start time	DIMPTIME	Optional	Optional
113	10	CHAR	Deploy Activation start date	DACTDATE	Optional	Optional
123	8	CHAR	Deploy Activation start time	DACTTIME	Optional	Optional
131	4	CHAR	Set Prefix. If not specified, will be set to **S**.	CNPRF	Optional	Optional

Table 1-21. TSB Layout (Continued)

Position	Length	Type	Description	JCL Keyword	API	JCL
135	10	CHAR	Release ID	RLSEID	Optional	Optional
145	1	CHAR	Set Change Type. If not specified, will be set to **S**.	CHNGTYPE	Optional	Optional
146	1	CHAR	Set Execution Status: **I** – Immediate processing (default) **H** – Held	EXECSTAT	Optional	Optional
147	10	CHAR	Set ID – Returned value	SETID	Optional	NA
157	4	INT	BASEID – Returned Value	BASEID	Optional	NA
161	1	CHAR	ABORTFLG – What to do with an error: **Y** – Abort option: Stops adding any further Tasks to the Set and leaves any previously added Tasks in the Set. The user will then have to manually remove the Tasks from the Set before re-running the operation. **C** – Cleanup option: Stops adding any further Tasks to the Set, removes any previously added Tasks from the Set, and deletes the Set. **N** – Continue option (default): Continues adding Tasks to the Set, ignoring any Tasks that fail to be added to the Set, and then executes the Set.	ABORTFLG	Optional	Optional
162	1	CHAR	VCTLFLG – Version Control Flag: **W** – Warning option: The version control check warning is heeded, and the Promote is failed. **N** – Continue option (default): The version control check warning is ignored, and the Promote continues even though the Setproc replacement prompt has a value of N (No).	VCTLFLG	Optional	Optional

Task Operation – Specify Tasks

Purpose

This call is only used in conjunction with a TASK BATOP call. It is used to specify individual Tasks to be put into the Set. One to many calls of this type follow a TASK BATOP call.

API Parameter List

ECIR, TSBT

The ECIR Data Structure must have the values in Table 1-22 set.

Table 1-22. ECIR Data Structure Values for Task Operation – Specify Task

Parameter	Value
OBJECT	TASK
METHOD	BATOPTSK

JCL Invocation

```
$DEFINE_TSBT
```

TSBT Data Structure

The TSBT data area contains the key fields needed to identify a Task. The TSBT layout is described in Table 1-23.

Table 1-23. TSBT Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	4	CHAR	Application ID of the Task	APPLID	Mandatory	Mandatory
5	4	CHAR	Component Type of the Task	MTYPE	Mandatory	Mandatory
9	4	CHAR	Component Class of the Task (default is **NULL**)	MCLAS	Mandatory	Mandatory
13	64	CHAR	Name of the Component	MNAME	Mandatory	Mandatory
77	246	CHAR	Relative Path of the Component (default is **NULL**, and this only applies to distributed components)	CRELPATH	Optional	Optional
323	10	CHAR	Set ID	SETID	Mandatory	NA*

* For the JCL interface, the Set ID is preserved from the TSB datagroup and does not need to be specified on this call. When using the API, the Set ID is required.

Task Operation – Delimit the Task Operations

Purpose

This call is only used in conjunction with a TASK LOAD (and any associated TASK LOADPART calls). It is used to delimit the loading of a Task and contains the User Extension Data. Every TASK LOAD call must be completed by a TASK LOADEND call.

API Parameter List

ECIR, TSBE

The ECIR Data Structure must have the values in Table 1-24 set.

Table 1-24. ECIR Data Structure Values for Task Operation – Delimit the Task Operations

Parameter	Value
OBJECT	TASK
METHOD	BATOPEND

JCL Invocation

```
$DEFINE_TSBE
```

TSBE Data Structure

The TSBE data area contains the Set ID. The TSBE layout is described in Table 1-25.

Table 1-25. TSBE Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	10	CHAR	Set ID of the Set that was returned from the TASK BATOP call.	SETID	Mandatory	NA*

***** For the JCL interface, the Set ID is preserved from the TSB datagroup and does not need to be specified on this call. When using the API, the Set ID is required.

Set Approvals

Purpose

This call is used to Approve or Deny a Set. The userID associated with the ECI calling Task or Job is used by ISPW for the security check.

API Parameter List

ECIR, STR

The ECIR Data Structure must have the values in Table 1-26 set.

Table 1-26. ECIR Data Structure Values for Set Approvals

Parameter	Value
OBJECT	SET
METHOD	APPRVMOD

JCL Invocation

```
$DEFINE_STR
```

STR Data Structure

The STR data area contains the fields necessary for the approval of a Set. The STR layout is described in Table 1-27.

Table 1-27. STR Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	10	CHAR	Set ID	SETID	Mandatory	Mandatory
11	10	CHAR	Approver	APPROVER	Mandatory	Mandatory
21	1	CHAR	Sequence	SEQ	Mandatory	Mandatory
22	1	CHAR	Approval Indicator: **A** – Approve D – Deny **P** – Pending (reset)	APRVCODE	Mandatory	Mandatory

Set Execute

Purpose

This call is used to release a Set for execution. It is valid only where the Execution Status of the Set is HELD. The userID associated with the ECI calling Task or Job is used by ISPW for the security check.

API Parameter List

ECIR, STR

The ECIR Data Structure must have the values in Table 1-28 set.

Table 1-28. ECIR Data Structure Values for Task Operation – Set Execute

Parameter	Value
OBJECT	SET
METHOD	EXECUTE

JCL Invocation

```
$DEFINE_STM
```

STM Data Structure

The STM data area contains the fields necessary for the release of a Set. The STM layout is described in Table 1-29.

Table 1-29. STM Layout

Position	Length	Type	Description	JCL Keyword	API	JCL
1	10	CHAR	Set ID	SETID	Mandatory	Mandatory
11	10	CHAR	Execute Code	EXECSTAT	Mandatory	Mandatory

Component Transport Control

Purpose

The TRNSPCTL AUTHREQ is used to send an operational request to a Component Transport Server. The requests can be used to initiate the following functions:

- Put the Component Transport Server into a hold state. When the server is in a hold state, all new requests will be queued and any warehouse datasets will be closed and de-allocated. This can be used to allow backups of warehouse datasets.
- Release the server from a hold state.
- Start the warehouse housekeeping function. If you are not running this function periodically by coding HKINTERVAL=0 in the Component Transport parameters, you can trigger the housekeeping through this call.

API Parameter List

ECIR, CTQ

The ECIR Data Structure must have the values in Table 1-30 set.

Table 1-30. ECIR Data Structure Values for Component Transport Control

Parameter	Value
OBJECT	TRNSPCTL
METHOD	AUTHREQ

CTQ Data Structure

The CTQ data area contains details required to perform the Component Transport function. The CTQ layout is described in Table 1-31.

Table 1-31. CTQ Layout

Position	Length	Type	Field	Description
1	4	CHAR	CTREQTYP	Request Type. Valid values are: **HOLD** – hold the server **RELE** – release the server **CWHK** – start warehouse housekeeping
5	2	BINARY	CTQCOMPT	Not used
7	8	CHAR	SCTSRVNM	Name of the Component Transport Server
15	4	CHAR	SWLBTYPE	Not used
19	254	CHAR	SWLBNAME	Not used
273	24	CHAR	SSTORKEY	Not used
297	8	CHAR	TCTSRVNM	Not used
305	4	CHAR	TWLBTYPE	Not used
309	254	CHAR	TWLBNAME	Not used
563	24	CHAR	TSTORKEY	Not used
587	24	CHAR	VSTORKEY	Not used
611	8	CHAR	ERRID	If the return code is not zero, this contains the ID of the error.
619	8	CHAR	ERRCODE	If the return code is not zero, this contains the error code.
627	72	CHAR	ERRSTXT	If the return code is not zero, this contains a short error message.
699	255	CHAR	ERRLTXT	If the return code is not zero, this contains the long error message.

Chapter 2.
Standalone Load Modules

The term *standalone load modules* refers to load modules that must be managed without any source. They are not generated; they are simply migrated to production. An example would be the production components supplied by vendors.

Environment Assumptions

It is assumed:

- DCB information is the same between the TEST, HOLD, and PROD level datasets.
- No link option is provided.

Production Cycle Processing

The following are the ISPW production cycle processes for standalone load modules.

Add Task

Standalone load modules, by definition, are not “created”. They are copied. Therefore, they must exist at the time the Add function takes place. The Extended Add function is used to specify the load dataset from which the module is to be copied. This load library name is stored in the &ALTLIB variable and is used in the copy step of the TEST level generate job as the “copy from” dataset. If a backup library has been specified for the Associated load module component in M.AD, it is permissible to specify ACTION = H when adding a load-only task to an assignment. &ALTLIB will be set to the backup load library.

Checkout

An empty source module is created and used for checkout control. The TEST Generate copies the load module from the library specified on the Add Task to the TEST level load library. The Generate for a HOLD level copies the load module from the previous level load library to the HOLD level load library.

Select

Load modules cannot be edited with the regular ISPF editor. A message identifying the dataset type as invalid will appear if this option is used.

Browse

The load module itself is browsed using the ISPF browse facility. This option can be used at any time.

Generate

The TEST Generate copies the load module from the library specified on the Add Task to the TEST level load library. The Generate for a HOLD level copies the load module from the previous level load library to the HOLD level load library.

Promote

The empty source module and load module is promoted to HOLD.

Regress

The empty source is copied back to the previous level. The load module is deleted, unless it has been defined to be retained in M.AD.

Note: The load module is ***not*** copied back to the previous level load library. If required or desired, other options can be "disallowed" by using the WZTLCS# panel as a pre-exit.

Initialization

M.CT Values

The values listed in Table 2-1 should be used for the M.CT entry.

Table 2-1. M.CT Values

Field	Value
Description	**Load-only modules (Load copied-no source)**
Technology	**LC**
Cycle Flag	**Y**
Generate Flag	**Y**
Generate Skeleton	Skeleton name for component type.
Generate Job	Name of the batch job to perform the demanded generate.
Generate Table	Name of the table to be used in the demanded job.
Test Gen. Panel	Test panel name for component type.
Hold Gen. Panel	Hold panel name for component type.
Prod Move Job	Currently not used.
Model	**DUMMY**

M.EX Values

The values listed in Table 2-2 should be used for the M.EX entry.

Table 2-2. M.EX Values

Operation	Field	Value
B	Exit Key	LC
	Description	(LOAD CONTROL): LOAD MODULES
	Pre-op	ISPEXEC DISPLAY PANEL(WZTLCB#)
	Post-op	ISPEXEC DISPLAY PANEL(WZTLCB#)

Table 2-2. M.EX Values (Continued)

Operation	Field	Value
C	Exit Key	LC
	Description	(LOAD CONTROL): LOAD MODULES
	Pre-op	ISPEXEC DISPLAY PANEL(WZTLCC#)
	Post-op	ISPEXEC DISPLAY PANEL(WZTLCC#)
G	Exit Key	LC
	Description	(LOAD CONTROL): LOAD MODULES
	Pre-op	ISPEXEC DISPLAY PANEL(WZTLCG#)
R	Exit Key	LC
	Description	(LOAD CONTROL): LOAD MODULES
	Pre-op	ISPEXEC DISPLAY PANEL(WZTLCR#)
S	Exit Key	LC
	Description	(LOAD CONTROL): LOAD MODULES
	Pre-op	ISPEXEC DISPLAY PANEL(WZTLCS#)

M.AD Values

Follow the procedure in Table 2-3 to set up the M.AD entries.

Table 2-3. M.AD Values

Step	Action
1	Add a component Type to an application in M.AD for the load libraries and define the load library names in the Names section. Example Names entries are shown in Figure 2-1 on page 2-4.
2	Add a component Type to an application in M.AD for the source libraries required for load-only processing. Example Name entries are shown in Figure 2-2 on page 2-4.
3	Add a Generate association for the LCTL component Type to indicate that a LOAD will be generated and to point to the component type of the (LOAD) libraries to be used. An example entry is shown in Figure 2-3 on page 2-4.
4	The flags in M.AD for the test level should be set up so that associations are moved to the HOLD level. However, at least the test level generate needs to be done so module has been copied. An example entry is shown in Figure 2-4 on page 2-5.

Names Entries for LOAD

Example M.AD Names entries are shown in Figure 2-1 for application component type LOAD.

Figure 2-1. M.AD Names Entries for LOAD

```
WZZMADD D/N       BLM LIBRARY NAMES TABLE (PROD)                BROWSE MODE
Command ===>                                                  Scroll ===> CSR

List Commands: L Locate Entry, U Update Mode

                   Lib    Library
Type Clas  Lev     Type   Name

LOAD       FIXH    PDS    BLM.FIXH.LOADLIB
LOAD       FIXT    PDS    BLM.FIXT.LOADLIB
LOAD       HOLD    PDS    BLM.HOLD.LOADLIB
LOAD       PROD    PDS    BLM.PROD.LOADLIB
LOAD       TEST    PDS    BLM.TEST.LOADLIB
****************************** Bottom of data ******************************
```

Name Entries for LCTL

Example M.AD Names entries are shown in Figure 2-2 for application component type LCTL.

Figure 2-2. M.AD Names Entries for LCTL

```
WZZMADD D/N       BLM LIBRARY NAMES TABLE (PROD)                BROWSE MODE
Command ===>                                                  Scroll ===> CSR

List Commands: L Locate Entry, U Update Mode

                   Lib   Library
Type Clas  Lev     Type  Name

LCTL       FIXH    PDS   BLM.FIXH.CHKOUT.EMPTYSRC
LCTL       FIXT    PDS   BLM.FIXT.CHKOUT.EMPTYSRC
LCTL       HOLD    PDS   BLM.HOLD.CHKOUT.EMPTYSRC
LCTL       PROD    PDS   BLM.PROD.CHKOUT.EMPTYSRC
LCTL       TEST    PDS   BLM.TEST.CHKOUT.EMPTYSRC
****************************** Bottom of data ******************************
```

Association Entry for LCTL

An example M.AD Associations entry is shown in Figure 2-3 for application component type LCTL.

Figure 2-3. M.AD Associations Entry for LCTL

```
WZZMADA D/A       BLM  ASSOCIATIONS TABLE (PROD)                UPDATE MODE
Command ===>                                                  Scroll ===> CSR

List Commands: A Add Entry, L Locate Entry, B Browse Mode
Line Commands: D Delete

                              Target              Assoc  Assoc  Assoc
    Type   Clas   Association   Env.   Set  Seq.   Appl   Type   Clas

    LCTL               G              LOAD  1     BLM    LOAD
****************************** Bottom of data ******************************
```

Flag Entries for LCTL

An example M.AD Flags screen is shown in Figure 2-4 for application component type LCTL.

Figure 2-4. M.AD Flags Screen for LCTL

```
WZZMADF D/F       SAMP CONTROL FLAGS TABLE (PROD)              BROWSE MODE
Command ===>                                                Scroll ===> CSR

List Commands: L Locate Entry, U Update Mode

                  Promote  Version   Generate   Implement    Pack Keep
Type Clas  Lev    Method   Control   Opt  Chk   Opt  Chk     Src  Memb

LCTL       FIXH     A         N
LCTL       FIXT     A                  R    Y
LCTL       HOLD     A         N
LCTL       PROD               N
LCTL       TEST     A                  R    Y
```

These flag settings will force a successful Generate to be done at the TEST levels, and then on the promotes, the LOAD will be moved up with the component. The Version Control flags are set to **N** in this case, because it represents the situation where the component versions are not checked out based upon another version, but instead delivered from an external source.

Site-specific Components

Generate Skeleton Requirements

The generate skeleton WZTLCG# is supplied in the ISPW base SAMPLIB dataset as a sample.

Maintenance Issues

Pre-Exit and Post-Exit Panels

The pre-exit and post-exit panels use `.RESP=ENTER` or `.RESP=END` in the `)INIT` section, so they are normally processed without being displayed.

The recommended approach for pre-exits and post-exits is to use the `&TECHNOL` value (`LC`) as the Exit Key, rather than the `&MEMBTYPE` value. Then, the value for `&MEMBTYPE` can be whatever is desired.

Load Library Blocksize

The IEBCOPY program will not copy to a smaller blocksize for RECFM=U, so ideally the TEST/HOLD/PROD load libraries must be set up with identical blocksizes, which would be equal to or greater than that for any load library which would be specified as &ALTLIB. If this is not possible, the COPYMOD control statement, with the appropriate parameters coded, may be used in an IEBCOPY step to “reblock” to a smaller blocksize for the output library.

IEBCOPY

The IEBCOPY program does not issue any MVS enqueue to attempt to ensure exclusive use for update of the dataset into which it is copying. Therefore, there might be instances when a Generate job would fail (ABEND S213-30) if it attempted to update a load library simultaneously with another job. “Replacement” programs are available (for example, PDSFAST) which can avoid this problem; alternatively, Compuware can supply an

Assembler program which runs in a TSO environment (foreground or background/batch) and issues an appropriate enqueue and dequeue around a call of IEBCOPY.

Note: If update activity on the TEST and HOLD load datasets is reasonably low, this problem may not be encountered frequently enough to be of real concern.

Linkage Editor

The only change required is to substitute link-edit JCL for the IEBCOPY in the Generate skeleton used. The advantage would be that the linkage editor uses the appropriate enqueue for its output dataset. A possible disadvantage might be that a user might need to specify correct link-edit parameters if they differed from those that were coded as the default in the skeleton.

Load Module Aliases

Although the example skeleton provided is not sufficient to properly handle load modules with aliases, this capability could be added without great difficulty.

Chapter 3.
DB2 Programs, Plans, and Packages

Often changes to programs that call DB2 (DB2 programs) have accompanying data definition language (DDL) changes. The coordination of both of these changes is difficult, but very important. This chapter focuses only on DB2 programs, plans, and packages.

Note: ISPW always performs a BIND on plans and packages—never a REBIND. **No REBIND option exists in ISPW.**

General Information

Intended Audience

You should have basic knowledge of DB2, including knowledge of database request modules (DBRMs), timestamps, plans, and packages.

DB2 Interface Overview

Processes are provided to construct and execute plan and package binds. The bind process can be driven by a Generate of a plan or a DB2 program, or can be requested independently.

Plan Bind Process

A plan bind may be required because of a change in a plan's attributes or because of a DBRM (DB2 program) update. Therefore, ISPW provides for initiating a bind by generating the plan task itself or indirectly by the generate/compile of the DB2 program task that uses the DBRM/plan Component Reference to determine which plans to bind.

Package Bind Process

A package bind is normally only required when the DBRM (DB2 program) is updated. Therefore, ISPW treats package attributes as an extension of generate parameters, and the package bind is incorporated as a step in the program generate processing. If a site continues to reference DBRMs directly in plan definitions, the DBRM to plan Component Reference repository can be utilized to automatically bind all plans affected by the DBRM change.

DB2 Interface Data Repository

There are several ISPW Repository objects to support the plan/package. The tables and entries required for the DB2 interface are described below and include:

- Plan Component Table
- Generate Parameters Table (<*appl*>CMPL)
- Application Definition (M.AD) Plan Implementation entries
- Application Definition (M.AD) Levels entries
- External Reference Table (M.ER)
- DBRM to Plan Component Reference Tables.

Plan Component Table

The plan component table is an ISPF table containing the bind attributes for an individual plan task (for example, the bind statement parameters). This library is defined in M.AD as the source library for the Plan components. In the bind process, the plan parameter table associated with the Task is used to build the bind statement executed.

Generate Parameters Table

This is an ISPF table maintained for each application containing information by Component Name and Type. It is used to control the generate processing of that component. A set of variables is stored on this table which indicate:

- whether a component uses DB2,
- whether a bind should be processed as part of the program generate,
- whether the component employs a DB2 package, and
- the variables for package attributes which may be used to override general specifications stored in the M.AD Plan Implementation entries for the package bind.

M.AD Plan Implementation entries

This is a part of M.AD containing information by application, level, and implementation "rule" to simplify the plan/package definition process while giving control and flexibility to the bind process. It supports situations where different databases or package collections need to be targeted at different levels and helps manage the mapping of attributes such as plan name, owner, qualifier, and other plan/package parameters which may vary in different levels or implementations of an application. For more information, see "AD (P) - Plans" in the chapter entitled "Maintenance Functions" in the *ISPW Technical Reference*.

M.AD Levels entries

This is a part of M.AD containing information by application and level that contains a number of DB2-related variables applicable to the level, such as:

- the DB2 subsystem used,
- the plan Component Reference table name,
- DBRM libraries, and
- the "rule" to be used in implementing plans and packages.

For more information, see "AD (S,L) - Levels" in the chapter entitled "Maintenance Functions" in the *ISPW Technical Reference*.

External Reference Table

This table defines installation-dependent variables and datasets to ISPW. For each DB2 subsystem, the DB2 load library name is defined here.

DBRM to Plan Component Reference Tables

Packages are viewed as the solution to the problems of DBRMs that are referenced in multiple plans. However, unless a site is utilizing packages exclusively, a mechanism is needed to identify and bind all plans that are directly referencing an updated DBRM. Therefore, ISPW continues to provide support for automating the bind of these plans using the DBRM to plan Component Reference repository. An ISPF table is maintained at each level to contain this impact information. As plan component types are processed through the update cycle, the DBRMs required by a plan are extracted and related to the plan/application.

Note: The CLIST WZU2TC# is provided to create empty Component Reference tables. Parameters to the CLIST are:

- `<table>` = table name,
- `<lib>` = R for PROD level, W for other levels.

Example: `EXEC %WZU2TC# DEVPLAN W.`

Environment Assumptions

- A plan or package is bound to only one subsystem for any level.
- All subsystems that may be accessed via bind processing of any bind job (either generate or promote) must be available on the same OS/390 system image that the bind job is executing on. If propagating binds, this will include those subsystems associated with any lower levels to be bound.

Production Cycle Processing

DB2 Program Tasks

A DB2 program task follows the standard processing outlined for generated type tasks. The additional processing required for DB2 generation is handled in the input parameters specified on the panels and the steps defined in the skeletons for the generate. Additional pre-exit processing occurs on "P-Style" promotes to bind the package and to bind those plans using the DBRM as indicated in the DBRM/plan component reference repository. This is accomplished by setting up the M.EX entry for each program Component Type using DB2 as specified above.

Note: Currently, for "P-Style" promotes, the HOLD DBRM is copied to the next level in the pre-exit before bind processing is performed. If some bind should fail, a non-zero return code will be returned from the pre-exit, standard processing will not be done, and the DBRM at the target level will be out of sync with the target level source/load until the problem is resolved.

Plan Tasks

A Plan task follows the standard processing outlined for generated type tasks with the exceptions listed in the following sections and additional processing controlled by M.EX entries.

Browse

In the pre-exit, the table of bind parameters is displayed. Regular browse processing is bypassed.

Checkout

In the regular processing, the plan is not actually copied, but the bind attribute table that identifies how the plan is bound is copied from production to TEST. If this is a new plan, the model is copied. In the post-exit, the plan Component Reference information for the module is updated in the TEST level with the PROD level information (or deleted when ACTION=D).

Select

In the pre-exit, the plan attribute table is updated using ISPF table services. The final step in the pre-exit updates the TEST level plan Component Reference information using the updated plan attribute table.

Promote

In the post-exit, the HOLD level plan Component Reference information for the module is updated (or deleted for ACTION=D).

In the pre-exit, the plan is bound at the target level using the plan attribute table of the current level. If the bind fails, a non-zero return code will be returned from the pre-exit and standard processing will not be done. Normal promote processing updates the target level plan attribute table. In the post-exit, the Component Reference information is updated using the plan attribute table (or deleted for ACTION=D).

Regress

In regular processing, the source is moved to the previous level and—depending on the setting of the Retain flag for the library Names—load and GMT modules are deleted. If the load is deleted, plan/package binds may need to be done for the levels affected. In the pre-exit, the previous level plan Component Reference table is restored with the information in the plan attribute table regressed.

Initialization

Defining the DB2 Environment

The key to installing the ISPW DB2 interface is defining the environment. Are the "rules" for implementing plans and packages defined at a high level (for the whole site) or a low level (for an application)? Ultimately, for each application and level, the following need to be identified:

- the subsystem,
- the DBRM libraries used,
- the owner and qualifier values,
- any plan and package naming conventions used, and
- any other plan/package bind parameters that may be required.

This environment information is stored in ISPW's central data repository and is described below.

M.CT Values

Two categories of Component Types are used to support DB2:

- Program Component Types, and
- Plan Component Types.

It is not necessary to set up specific Types for DB2 programs versus non-DB2 programs, because use of DB2 can be indicated on the generate panel. The refer-only Component Type of DBRM is required for the DBRM generated for DB2 programs.

M.CT Values for Program Types

The values in Table 3-1 show a sample M.CT entry for a DB2 program.

Table 3-1. M.CT values for Program Types

Field	Value
Description	**DB2 Cobol program**
Cycle Flat	**Y**
Generate Flag	**Y**
Generate Skeleton	Skeleton name for component type.
Generate Job	Name of the batch job to perform the demanded generate.
Generate Table	Name of the table to be used in the demanded job.
Test Gen. Panel	Test panel name for component type.
Hold Gen. Panel	Hold panel name for component type.
Prod Move Job	Currently not used.

M.CT Values for Plan Types

The values in Table 3-2 show a sample M.CT entry for a DB2 plan.

Table 3-2. M.CT values for Plan Types

Field	Value
Description	**DB2 Plan bind attributes table**
Cycle Flag	**Y**
ISPW Typed	**PLAN**
Generate Flag	**Y**
Generate Skeleton	Skeleton name for component type.
Generate Job	Name of the batch job to perform the demanded generate.
Generate Table	Name of the table to be used in the demanded job.
Test Gen. Panel	Test panel name for component type.
Hold Gen. Panel	Hold panel name for component type.
Prod Move Job	Currently not used.
Model	$PLAN from the ISPW SAMPLIB (a model is required).

M.CT Values for DBRM Types

The fields and values in Table 3-3 show a sample M.CT entry for a DBRM.

Table 3-3. M.CT values for DBRM Types

Field	Value
Description	**DB2 Database Request Module**
Cycle Flag	**Y**
Prod Move Job	Currently not used.
Execution Env.	**REFR**

M.EX Values

M.EX Values for Program Exits

The values in Table 3-4 should be used for the M.EX entry for programs.

Table 3-4. M.EX values for Program Exits

Op	Field	Value
P	Exit Key	The component type
	Description	**DB2 Bind of plans/packages**
	Pre-op	ISPEXEC SELECT CMD(%WZU2PP#P &DEBUG)

M.EX Values for Plan Exits

If Using Packages and Collections

The values in Table 3-5 should be used for the M.EX entries if using packages and collections.

Table 3-5. M.EX values for Plan Exits

Op	Field	Value
B	Exit Key	PLAN
	Description	DB2 exit to browse plan attribute table
	Pre-op	ISPEXEC SELECT CMD(%WZU2PB# &DEBUG)
S	Exit Key	PLAN
	Description	DB2 exit to update plan attribute table
	Pre-op	ISPEXEC SELECT CMD(%WZU2PS# &DEBUG)

If Using DBRMs Directly with Plans

The values in Table 3-6 should be used for the M.EX entries if using DBRMs directly with plans.

Table 3-6. M.EX values for Plan Exits

Op	Field	Value
C	Exit Key	PLAN
	Description	DB2 exit to update XREF table info
	Post-op	ISPEXEC SELECT CMD(%WZU2PC# &DEBUG)
D	Exit Key	PLAN
	Description	DB2 exit to restore XREF table info
	Pre-op	ISPEXEC SELECT CMD(%WZU2PD# &DEBUG)
P	Exit Key	PLAN
	Description	DB2 exit to bind plan/update XREF table info
	Pre-op	ISPEXEC SELECT CMD(%WZU2PP#O &DEBUG)
	Post-op	ISPEXEC SELECT CMD(%WZU2PP# &DEBUG)
R	Exit Key	PLAN
	Description	DB2 exit to update XREF table info
	Post-op	ISPEXEC SELECT CMD(%WZU2PC# &DEBUG)

Table 3-6. M.EX values for Plan Exits (Continued)

Op	Field	Value
X	Exit Key	PLAN
	Description	DB2 exit to restore XREF table info
	Pre-op	ISPEXEC SELECT CMD(%WZU2PX# &DEBUG)

M.AD Levels

For detailed descriptions of the fields shown in Figure 3-1, see "AD (S,L) - Levels" in the chapter entitled "Maintenance Functions" in the *ISPW Technical Reference*.

Figure 3-1. M.AD/L (Levels)

```
ISPW M.AD/L       BROWSE BLM  LEVEL DETAIL (PROD)
Command ===>

Level  (KEY) ==> TEST     (TEST HOLD PROD, etc)
 Next Level  ==> HOLD
 Impl Exit?  ==>          (Y - Yes)

 Set Scheduling Information:
   Set Class      ==>    Job Name ==>         Queue Name ==>
   Failure Notify ==>

 DB2 Information:
   Impl Name/Rule ==> APPL     Name or Rule to determine Plan Implementation
   DB2 Subsys     ==> DB2T     Sub-system applicable for this Level
   DBRM Libs      ==> CORE.TEST.DBRMLIB

   XREF Name      ==> TEST$PLN
   XREF Lib       ==> W        R - Read only, W - Read/Write

Press END to return
```

M.AD Plan Implementation

The specifications shown in Figure 3-2 drive the bind statement generation of plans and packages. There may be more than one implementation at a given level. For detailed field descriptions, see "AD (P) - Plans" in the chapter entitled "Maintenance Functions" in the *ISPW Technical Reference*.

Figure 3-2. Modify BLM Plan Implementation Detail (Prod)

```
WZZMADP1 /P       MODIFY BLM PLAN IMPLEMENTATION DETAIL (PROD)
Command ===>

Enter required details:

 Level  (KEY) ==> TEST     (TEST HOLD PROD, etc)
 Impl   (KEY) ==> BLM      (Implementation name)
 Do BINDs?    ==> N  (Y/N) N=No bind processing for PLANs or PACKAGEs
 Owner        ==> BLMDEV           Qualifier    ==>
 Subsystem    ==> DB2T             Explain      ==> Y         (Y/N)

 Logical  Collection IDs ===> BLMCOLL
 Physical Collection IDs ===> BLMD

Plan Bind Details :
 Name Prefix  ==> D       Name Suffix ==>
 Other Parms  ==>

Package Bind Details :
 Collection   ==> BLMD
 Other Parms  ==>

Press ENTER to complete the change or END to terminate
```

M.AD Associations

The associations values for a DB2 program Type might be defined as shown in Figure 3-3.

Figure 3-3. BLM Associations Table (PROD)

```
WZZMADA D/A       BLM ASSOCIATIONS TABLE (PROD)                  UPDATE MODE
Command  ===>                                                 Scroll ===> CSR

List Commands: A Add Entry, L Locate Entry, B Browse Mode
Line Commands: D Delete

                              Target              Assoc  Assoc  Assoc
    Type   Clas   Association   Env.   Set  Seq.   Appl   Type   Clas

    COB                C               LKED  1     BLM    LKED
    COB                G               GMT1  1     BLM    DBRM
    COB                G               LOAD        BLM    LOAD
    COB                I               DCL   1     BLM    DCLG
    COB                I               MAC   1     BLM    COPY
    COB                I               SYS   1     BLM    LOAD
    COB                I               SYS   2     BLM    LODS
    COB                R                           BLM    LKED
```

M.ER Values

As shown in Figure 3-4, for each DB2 subsystem, an M.ER entry is required to describe the DB2 load library. The Type Code must be of the format: `DB2<subsystemid>`.

Figure 3-4. M.ER — Browse External Reference Detail (PROD)

```
ISPW M.ER          BROWSE EXTERNAL REFERENCE DETAIL (PROD)
Command ===>


  Type Code (KEY) ==> DB2DB2T            View Code (KEY) ==>

  Description     ==> DB2 LOAD LIBRARY FOR SUBSYSTEM: DB2T

  Variable/Dsname ==> SYS1.DB2T.DB2.LOADLIB
  (Fully qualified if data set name, no quotes)

Press END to return
```

Example Generate Panel

An example generate panel is shown in Figure 3-5, with the flags indicating whether the program uses DB2.

Figure 3-5. Example Generate Panel

```
ISPW P/S/G  ------------ Generate Options -----------------------------------
COMMAND ===>

 Type ==> C            Name(s) for Generated Type(s):
 Name ==> WZZRPCR      Src1 ==> WZZRPCR   Src2 ==> WZZRPCR   Src3 ==> WZZRPCR
 Load ==> WZZRPCR      Lod1 ==> WZZRPCR   Lod2 ==>           Lod3 ==>
 Program (Y/N) ==> N          SQL used?   ==> Y  Package used?     ==> Y
 Do Binds?     ==> Y  (Y/N/B)  Bind Plans? ==> N  Prod DB2 Subsytem ==>
 Target Env.      ==>         (Blank or MSP)
 Add'l Gen. Opts  ==>
 Add'l Link Opts  ==>



 SAVE changes to above Options? ==> Y  (Y/N)                Edit JCL?   ==>
 Include TEST: Copylib? ==> Y  Linklib? ==> Y    (Y/N)      Panel Lock? ==> N
 DISPLAY/CHANGE DB2 Package Bind Parameters? ==> N  (Y/N)
 JOBCARD Information:
 ==> //&LOADNAME  JOB  (,),'ISPW',CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
 ==> //        NOTIFY=KARYNS
 ==> //*
 ==> //*
     ----|----1----|----2----|----3----|----4----|----5----|----6----|----7--
 Press ENTER to Submit Generate or  END to terminate
```

Example Bind Options Panel

If the **DISPLAY/CHANGE DB2 Package Bind Parameters** field has been set to **Y**, then the panel WZXGENP shown in Figure 3-6 is displayed.

Figure 3-6. DB2 Package Bind Option Screen

```
ISPW P/S/G --------------  DB2 PACKAGE BIND OPTIONS   ------------------------
COMMAND ===>


 Collection   ==>

 Owner        ==>
 Qualifier    ==>

 Explain      ==> N         (Y/N)
 Validate     ==>
 Release      ==>
 Isolation    ==>
 Flag         ==>
 Currentdata  ==> N         (Y/N)
 Degree       ==>

 Press ENTER to Submit Generate or  END to terminate
```

Site-specific Components

Introduction

Each site must build customized generate panels and skeletons according to their requirements and guidelines. For reference purposes, sample components are supplied in the ISPW installation SAMPLIB dataset.

DB2 Program Processing

SAMPLIB components which may be reviewed to aid in the setup of DB2 program processing include those listed in Table 3-7.

Table 3-7. DB2 Program Processing SAMPLIB

Type	Name	Description
Panel	WZUGD	TEST level generate example
Panel	WZUGH	HOLD level generate example
Skeleton	WZUG	Generalized mainline “driver” skeleton
Skeleton	WZU@JOB	Jobcard
Skeleton	WZU@MTYP	Sets parameters by Component Type
Skeleton	WZU@DB2	DB2 pre-compile step
Skeleton	WZU@COB	COBOL compile step
Skeleton	WZU@LKED	Link-edit step
Skeleton	WZU@DBRM	Copy of temp DBRM to PDS
Skeleton	WZU@DB2B	DB2 Bind processing
Skeleton	WZU@INPR	Reset task “in process” status

DB2 Plan Processing

SAMPLIB components which may be reviewed to aid in the set up of DB2 plan processing include those listed in Table 3-8.

Table 3-8. DB2 Plan Processing SAMPLIB

Type	Name	Description
Panel	WZUGD2	TEST level generate example
Panel	WZUGH2	HOLD level generate example
Skeleton	WZUGBIND	Mainline skeleton
Skeleton	WZU@JOB	Jobcard
Skeleton	WZU@DB2B	DB2 Bind processing
Skeleton	WZU@INPR	Reset task “in process” status

Jobcard Changes for Binds

If DB2 jobs must run under special job classes, be sure the analysts know to specify this class on their jobcard when requesting DB2 services. If the site generates jobcards for the analysts, check for the ISPWTYPE=PLAN and check for DB2 program Component Types (if program generates have a bind step) to set the proper job class values.

Plan Attributes ISPF Table Model

A model is required for every new Plan task. It is required because a Plan task is an ISPF table and is processed using ISPF table services. Plan Component Types must specify a model in M.CT, which may be overridden in M.AD for specific applications if they require a different model. A model Plan attributes ISPF table is supplied in member $PLAN found in the ISPW installation SAMPLIB dataset. Copy this model to the site models dataset and reference this dataset member in M.CT for all Plan Component Types.

The model provided is very simple. A site may wish to add additional parameter/value rows into the site Plan model. If so, use ISPF table services to update the model.

Other Issues

DB2 Security

The binding of packages and plans requires that certain DB2 privileges are granted. At the TEST level where individual users need to bind plans and packages, it might be necessary to give them these privileges for each of the Test “DB2 Owners” they are concerned with. At the HOLD levels, DB2 Binds are either performed by the Set Processor ID or the Demanded Batch job ID (depending upon how the generate was submitted). These IDs should be regarded as secure, and therefore it is safe to grant more system-wide privileges to them (for example, BINDAGENT). At the PROD level, only the Set Processor ID requires these privileges.

Minimizing Timestamp Errors with Bind Propagation

The ability to “propagate” binds has been incorporated into the DB2 interface.

For example, suppose a site concatenates load libraries for execution (TEST/HOLD/PROD) and deletes the load and DBRM modules as a task is promoted through the cycle. Then execution of a DB2 program will fail at the TEST level after that task has been promoted to HOLD until the plan and/or package is bound on the TEST subsystem using the DBRM in the HOLD level library.

This out-of-sync situation can be minimized by using the propagation feature to execute additional binds for lower levels in a bind request. This is achieved by passing the appropriate values for the following propagation parameters to the bind routine WZU2P#B.

PROPAGATE

Controls whether propagation is performed.

- PROPAGATE (N) – No propagation (default)
- PROPAGATE (Y) – Propagation is done.

PROPCNTL

Controls the scope of the propagation by promotion path determined by the start level indicated.

- PROPCNTL (0) – Propagate down the current path (default)
- PROPCNTL (1) – Propagate down the "other" paths
- PROPCNTL (ALL) – Propagate down all paths.

PROPEXCL....CONTINUE HERE

Controls binding of the level used to determine the scope of the propagation.

- PROPEXCL (N) – Bind the current level (default).
- PROPEXCL (Y) – Exclude the current level from bind processing.

Note: Since the load and DBRM are copied from the final HOLD level to the PROD level, propagation is not normally required because the timestamp values are identical. If propagation was done for the HOLD bind, the lower levels in the path were also bound using that DBRM. If propagation is required to process levels in other paths or other scenarios, it is strongly recommended that this be handled in a post-promote process. Then only the failure of a bind for the PROD level will stop the move of a task.

Startup Issues

For sites already using DB2 and implementing ISPW's DB2 component management, it is advisable to construct a load process that reads the DB2 catalogs to build both the individual plan bind parameter tables and the DBRM/plan impact Component Reference repository. The impact information must also identify each plan to an ISPW application. Otherwise, every plan must be moved through the circuit to production. Sites just starting to use DB2 are not greatly affected.

DB2 Package Version Management

ISPW's DB2 Package Version Management is configurable by Application and Level. This allows users to flexibly manage the DB2 Package Versions in the DB2 catalog.

It is assumed that the reader has a basic knowledge of DB2. This includes knowledge of database request modules (DBRMs), timestamps, plans, and packages.

ISPW Services

Some local configuration is required to enable this option. If it is preferred to perform this work externally, ISPW Services can be arranged.

Background

DB2 has a feature that allows multiple versions of a DB2 Package to exist. It is easy to create Package Versions because all it requires is an option on the DB2 Pre-compile. The challenge, however, is in having a process that FREEs old, unused versions at the appropriate time.

ISPW's new Package Version Management:

- tracks the Package Versions created within ISPW, and
- provides a Housekeeping mechanism for inactive versions to be freed automatically.

Overview

ISPW provides a version ID to the DB2 pre-compiler/in-line compiler that allows an easy correlation between the program version managed by ISPW and the DB2 packages created for that version.

DB2 Packages will become eligible for clean-up (with a DB2 FREE PACKAGE command) when their program versions are deleted in ISPW or become "Historical". The Package Version Housekeeping can be configured in each ISPW environment with rules that specify when versions are to be freed.

Internal Package Tracking

When Package Version Management is enabled for a level, any DB2 Bind performed as part of the ISPW process will be tracked in an internal ISPW repository table. Details of the Bind are matched to the ISPW Task that is the source of the DBRM being bound.

Housekeeping Process

A Housekeeping Process periodically runs which makes a call to ISPW and returns the details of any DB2 Packages that can be freed for that particular subsystem.

Installation

Supplied Components

Table 3-9 lists the components that are supplied for the Package Version Management feature.

Table 3-9. Package Version Management Supplied Components

Name	Type	Description
DDLW420E	SAMP	DDL for the creation of the new DB2 table WZT_PKG_BIND_HIST
BNDW420E	SAMP	Bind statement for a new ISPW DB2 Package WZZRPBN
WZZSC	LOAD	Replacement Load Module
WZZSP	LOAD	Replacement Load Module
WZZSM	LOAD	Replacement Load Module
WZZPR	LOAD	Replacement Load Module
WZZTS	LOAD	Replacement Load Module
WZZBP	LOAD	Replacement Load Module
WZZEXPC	LOAD	New Precompile wrapper
WZZERISPW	LOAD	Replacement Load Module

Table 3-9. Package Version Management Supplied Components (Continued)

Name	Type	Description
WZZRPBN	DBRM	New DBRM
WZZMADL1	PANL	Replacement M.AD(L) Panel
WZZMADL#	CLST	Replacement M.AD(L) driver routine
WZU2P#B	CLST	Replacement Bind Processor
WZU2P#BB	CLST	Replacement Batch Bind Processing Driver
WZU2P#PP	CLST	Replacement Package Bind routine
WZU@DB2F	CLST	New Package Free Routine
WZU@DB2V	CLST	New Package Free Verify Routine
WZU@PKGH	SAMP	New Sample Housekeeping Routine
WZU@DB2	SAMP	Replacement DB2 Pre-compile skel
WZU@BLD	SAMP	Replacement Build Register skel

Step 1. Add New DB2 Table

The DBA will need to:

- update the SAMP member DDLW420E to conform to the ISPW Repository naming standard, and
- create the new ISPW Repository DB2 Table using this updated SAMP member.

Step 2. Implement New Base Software

Copy the Base Software (types LOAD, DBRM, PANL, and CLST) to the correct libraries. Note that the CLST members might have already been customized in the SITE Application. If so, they need to be compared and updated as usual for an ISPW fixpac.

Step 3. Bind New DB2 Package

Update the SAMP member BNDW420E to conform to the ISPW Repository naming standard. Run this Bind statement to create the new ISPW DB2 Package WZZRPBN.

Step 4. Replacement SKELs

The SAMP members WZU@DB2 and WZU@BLD would have been customized in the SITE Application. These replacements need to be compared and updated as usual.

Step 5. Pre-compile Process

A new ISPW program is used instead of the standard DB2-supplied program for the pre-compile process (in skel WZU@DB2). This new program provides a variable containing an ISPW Generated Version that is suitable for the DB2 Pre-compile step. The format of the version ID is dependent upon what was specified in M.AD(L). This generated version ID is stored against the ISPW Task so that it can be used to determine if any subsequent DB2 Packages can be freed.

Step 6. Housekeeping Job

The SAMP member WZU@PKGH contains JCL to call the routine WZU@DB2F to free the DB2 Packages. This needs to be customized and made available for use. A copy of this JOB can be made which calls WZU@DB2V instead of WZU@DB2F. This copy will be useful in testing whether the interface is set up and working correctly.

Configuration

M.ER

Define a new M.ER variable called DB2PKG and set the value to **Y**. Refresh the server.

M.AD(L)

Once the interface is turned on, specification of the pre-compile version format must be done in M.AD(L) as shown in Figure 3-7.

Figure 3-7. Specifying Pre-compile Version Format in M.AD(L)

```
ISPW M.AD/L            MODIFY APPLICATION F42M STREAM F42M LEVEL (QA)
COMMAND ===>
Enter required details:

 Level  (KEY) ==> QA     Promote Analysis  ==>    (Y - Yes)
 Next Level   ==> PRD    Impact Approvals  ==>    (Y - Yes)
                         Impl Exit?        ==> D  (D - Deploy, Y - External)

 Warehouse for Sources  : Name ==> ISPWPROD   Policy ==> KEEP3INASSIGN
              Gen Types: Name ==> ISPWPROD   Policy ==> KEEP3INASSIGN

 Set Scheduling Information:
   Set Class      ==> A  Job Name ==>            Queue Name ==>
   Failure Notify ==>

 DB2 Information:
   Impl Name/Rule ==>          Name or Rule to determine Plan Implementation
   DB2 Subsys     ==> DSN1     Sub-system applicable for this Level
   DBRM Libs      ==> ISPW.F42M.QA.DBRM

   PKG Mgmt       ==> *         (* to enable)     Version ID ==> B (B - Build)
Press ENTER to complete the change or END to terminate
Note: You can add a new entry by overtyping the Key with a new unique value
```

The Package Management fields are shown in Figure 3-7. Table 3-10 lists valid values and their meanings.

Table 3-10. Package Management Values and Descriptions

Field	Description
PKG Mgmt	Blank – no management for this level ***** – Package Management enabled for this level
Version ID	**B** – Build ID

Precompile Process

The pre-compile skeleton requires changes to support the passing of the ISPW-generated version ID. Because of JCL and input parameter constraints, a new ISPW program wraps around the DB2 Pre-compiler.

ISPW Wrapper

The Pre-compile is called by the ISPW program WZZEXPC shown in Figure 3-8, which calls the standard DSNHPC, passing it the normal parameters. An ISPW-defined version ID is dynamically specified in the JCL and passed into the pre-compile when it is called.

Figure 3-8. ISPW Wrapper

```
//PC  EXEC PGM=WZZEXPC,
)SEL ~COND NE ~Z
//    COND=(~COND),
)ENDSEL
//    REGION=4M,
//    PARM='DSNHPC/HOST(~PCHOST),APOST,SOURCE,XREF,DATE(JIS),TIME(JIS)'
//STEPLIB  DD DISP=SHR,DSN=~WZZLLIB
//         DD DISP=SHR,DSN=~WTDB2LOD
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD DISP=(MOD,PASS),DSN=&&LISTING,
//            #ALLOTMP,
//            SPACE=(TRK,(30,15)),
//            DCB=(RECFM=VBA,LRECL=240,BLKSIZE=3120)
//SYSTERM  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  SPACE=(CYL,(5,2)),#ALLOTMP
//SYSUT2   DD  SPACE=(CYL,(5,2)),#ALLOTMP
//WZZBLDID DD  DISP=(OLD,PASS),DSN=&&BPBLD
//DBRMLIB  DD  DISP=(,PASS),DSN=&&DBRM(~GMT1NAME.),
//             #ALLOTMP,
//             SPACE=(CYL,(5,2,1)),
//             DCB=(DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=6160)
//SYSCIN   DD  DISP=(MOD,PASS),DSN=&&DSNHOUT,
//             #ALLOTMP,
//             SPACE=(TRK,(30,15),RLSE)
```

Bind Process

An enhancement to ISPW's Bind process calls new ISPW messages to record and retrieve information about the Bind being performed. As a result of this, ISPW has all the information it needs to know what Binds have been performed against which Tasks in any DB2 subsystem.

BINDHIST ADD

This new ISPW call is made in the Bind processing logic after a successful Bind to register the Package Version. Because the Bind Processing is often customized at a site, the code in Figure 3-9 is a REXX example of a successful call. In the standard ISPW-delivered sample, it would reside in WZU2P#PP.

Figure 3-9. BINDHIST ADD

```
If DB2PKGV = '*' Then Do
  p = cmd_output.0
  pkgname = cmd_output.p
  parse var pkgname pkgloc '.' pkgcoll '.' pkgname '.' pkgver
  DB2SSID = ssid
  PKGCOLL = collid
  PKGNAME = dbrm
  Parse var PKGVER '(' pkgverid ')'
  PKGBPARM = pkgbparm
  APPLID = applid
  STRMNAME = wtstrm
  CLVL = level
  If BINDLVL <> "" Then Do
     CLVL = BINDLVL
     End
  IMPLRULE = impl
  TASKIDX = substr(pkgver,4,12)
  BUILDIDX = substr(pkgver,16,12)
  If db2pkmgt = 'Y' Then Do
    DPENV = db2env
    SUBENV = db2sub
    SYSTNAME = db2syst
    End
  Else Do
    DPENV = ''
    SUBENV = ''
    SYSTNAME = ''
    End
  RETC = WZZTSI("BINDHIST" ,"ADD")
  End
```

BINDHIST STATUPD

In a Fallback (FB) Operation, it is necessary to update the Status of the Package in the Repository Table to indicate that it is active again (and update the previous one to Inactive). This is done with a call to ISPW that must be placed in the normal Package Bind process. The code will check to see if a corresponding Package exists and, if so, will call the function to update its status.

Figure 3-10. BINDHIST STATUPD

```
/* Process FB - CHECK PACKAGE VERSION MANAGEMENT */
/* If a Package exists then we don't need to Bind*/
"ISPEXEC VGET (OP DB2PKMGT DB2PKVER DB2ENV DB2SUB DB2SYST)"
If db2pkmgt = 'Y' & op = 'FB' Then Do
   DB2SSID = ssid
   PKGCOLL = collid
   PKGNAME = dbrm
   PKGVERID - DB2PKVER
   RETC = WZZTSI("BINDHIST" ,"GET")
   If retc = 0 & bndhstat <> 'H' Then Do
     bndhstat - 'A'  /* pkg exists - flip to active */
     RETC = WZZTSI("BINDHIST" ,"STATUPD")
     log = 'WZU2P#PP 'dbrm': Package exists(FB) - no bind'
     rc=logmsg(log)
     Iterate
     End
   End
```

The code in Figure 3-10 is placed in the bind processing logic and is called prior to the Bind. If the specific Package Version exists, the status update call is made and it will not need to do a new Bind. Because the Bind Processing is often customized at a site, the above code is a REXX example. In the standard ISPW-delivered sample, it would reside in WZU2P#PP.

Housekeeping Process

A new batch Housekeeping process (Figure 3-11) is supplied that issues DB2 FREE PACKAGE for any Package Versions that are identified as not being in use and able to be freed. This housekeeping job needs to be defined for each individual DB2 subsystem and scheduled to run periodically.

Figure 3-11. Housekeeping Process

```
//WZDB2PKG JOB ,,CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
//    NOTIFY=CRAIG
//* REUSE THE RX HARNESS TO INVOKE THE DB2 PKG FREE PROCESS
//* PARM1 = DB2 SUBSYSTEM
//* PARM2 = INACTIVE KEEP DAYS
//* PARM3 = DELETE KEEP DAYS
//WZZRX    EXEC PGM=IKJEFT01,REGION=0M,TIME=1440,DYNAMNBR=99,
//  PARM='%WZURX##I DSN1 9 2 WZU@DB2F ISPF'
//SYSPROC  DD DISP=SHR,DSN=ISPW.F42.SITE.PROD.CLIST
//         DD DISP=SHR,DSN=ISPW.W42.PRD.CLIST
..
..
```

Housekeeping Job Input Parms

The Input Parms for the Housekeeping job are listed in Table 3-11.

Table 3-11. Housekeeping Job Input Parms

Field	Description
PARM1	DB2 Subsystem – Specify the DB2 Subsystem in which housekeeping is to occur.
PARM2	Inactive Keep Days – The number of days after which an Inactive ISPW Task's Package will be freed.
PARM3	Delete Keep Days – The number of days after which a Deleted ISPW Task's Package will be freed.

The value specified for PARM2 needs to be carefully considered in the context of the organization's requirement for fallback. It should be complementary with the period of time that Superseded Backup tasks are kept in the warehouse and can be fallen back to. Check the relevant Warehouse Policy for this Application Level to confirm. If a Package Version is Freed and its Inactive Task is still backed up in the Warehouse, a subsequent Fallback operation will need to perform a Bind, rather than reinstate that Package Version, because it is no longer in the DB2 catalog.

The value specified for PARM3 can be set to a relatively small number of days, because the corresponding load and dbrm does not exist in ISPW after the associated Task has been deleted.

Processing Details

This section explains what occurs during normal processing when Package Version Management is enabled.

DB2 Table for Bind Details

A new ISPW DB2 Table (WZT_PKG_BIND_HIST) is used to store the details of successful Binds and then used in conjunction with the status of associated Tasks to determine what Packages might be available for FREE. To understand the process, it is worth looking at this table in detail, because when implementing this feature and testing it in the Customer environment, it will be necessary to inspect the contents of the table to make sure it has been set up correctly.

An example row with all of the columns is shown in Figure 3-12.

Figure 3-12. WZT_PKG_BIND_HIST DB2 Table Row Example

```
TABLE WZZF420.WZT_PKG_BIND_HIST                                  Format SNG
                                  Top Line is 1    of 27    in Row 1
Key    Column           Data
 U     DB2_SSID         DSN1
 U     PKG_COLLID       F42MDEV
 U     PKG_NAME         TEST99
 U     PKG_VERS_ID      B#7DDBDCC8CDCC7DDBE4439592
 U     BASE_ID                  1
       PKG_BIND_PARMS    OWNER(ISPWDEV) CURRENTDATA(NO) ISOLATION(CS).......
                        ..................................................
                        ..................................................
                        ..................................................
                        ........
       APPL_ID          F42M
       STREAM_NAME      F42M
       LVL              DEV1
       IMPL_RULE
  N    TASK_ID          '..H..
       TASK_LVL         DEV1
       BUILD_ID         '.U.nk
       SEC_TASK_ID
       DP_ENV
       SUB_ENV
       SYST_NAME
       CREATED_TSTAMP   2013-11-28-11.03.44.828984
       WS1
 U     CURR_ISPW_BIND   Y
       STATUS           H
       STATUS_TSTAMP    2013-11-28-11.49.49.323218
       FREE_TSTAMP      2013-11-28-11.49.49.323218
***  End of record  ***
```

The columns in WZT_PKG_BIND_HIST are described in Table 3-12.

Table 3-12. WZT_PKG_BIND_HIST DB2 Table Column Descriptions

Column	Description
DB2_SSID	DB2 Subsystem ID related to the Bind
PKG_LOC	Location ID for the Bind. Must not be blank. Default is the current Location.
PKG_COLLID	Collection ID for the Bind
PKG_NAME	Package Name for the Bind
PKG_VERS_ID	ISPW-defined version ID that was used during the Pre-Compile
BASE_ID	Internal ISPW Use
PKG_BIND_PARMS	Package Bind Parameters supplied to the Bind Process

Table 3-12. WZT_PKG_BIND_HIST DB2 Table Column Descriptions (Continued)

Column	Description
APPL_ID	Foreign key for the M.AD(P) entry used for the bind
STREAM_NAME	
LVL	
IMPL_RULE	
TASK_ID	Internal ISPW Task_ID identifying the Task for which the Bind was performed
TASK_LVL	ISPW Level the Task was at when the Bind was performed
BUILD_ID	Internal ISPW Use
SEC_TASK_ID	Internal ISPW Use
DP_ENV	Deploy Information if Bind performed in deploy processing
SUB_ENV	
SYST_NAME	
CREATED_TSTAMP	Timestamp when the Bind Row was added. (Should correspond to just after Bind Timestamp in Catalog.)
WS1	Internal ISPW Use
CURR_ISPW_BIND	Indicates whether this Bind is the deemed to be the current bind for the Task (value **Y**). A subsequent Bind will result in a new row, and this row will be marked **N**.
STATUS	Values can be: • Space – active • **I** – Inactive • **D** – Deleted • **H** – Historical
STATUS_TSTAMP	Timestamp of the most recent status change
FREE_TSTAMP	Timestamp when the FREE was performed

Bind Processing

After a successful bind, the new BINDHIST ADD call is made to ISPW to register the Bind in the DB2 Table. After it is successful, a row will have been added to the DB2 Table, which will relate the DB2 Package to the ISPW Task for which the Package was bound. These status values determine whether the DB2 Package is a candidate for a FREE when the housekeeping job executes. The status values for a new row will be as listed in Table 3-13.

Table 3-13. New DB2 Row Status Values

Column	Description
CURR_ISPW_BIND	**Y**
STATUS	<space>
CREATED_TSTAMP	Current date and time
STATUS_TSTAMP	Current date and time
FREE_TSTAMP	null

As the ISPW Task is promoted, generated, regressed, or deleted within ISPW, the status of the DB2 Package may change, depending upon whether the Task is re-generated, made

historical, or another bind performed. The DB2 Table will be updated to reflect that status as described in Table 3-14.

Table 3-14. Result of DB2 Table Update

Action	Outcome
Another Bind Performed	The PKG_VERS_ID has not changed. A new row is added, and the previously current row for the same PKG_VERS_ID will have the CURR_ISPW_BIND flag set to **N**, the STATUS set to **H**, and the STATUS_TSTAMP set.
Generate followed by Bind	A new PKG_VERS_ID will be generated, and so a new Row for the new Version will be added.
Any P, X or D operation where a Task is made Inactive or Deleted	If a Task with a corresponding row in the Bind History Table becomes inactive, then the row in the Bind History table will be set to **I**. This reflects the fact that the associated DBRM/LOAD is stored in ISPW and available in a fallback scenario. If a Task with a corresponding row in the Bind History Table is deleted or becomes inactive with Superseded Gone status, then the row in the Bind History table will be set to **D**. This reflects the fact that the associated DBRM/LOAD is deleted from ISPW.
A Free is performed by ISPW Housekeeping	The Package Version rows that are FREEd will be updated to a status of **H** and will now be there for the audit history.

Fallback Processing

One of the advantages of keeping Package Versions is that during a Fallback Operation, if a Package exists that corresponds to the new Load Module being replaced, then the Program will work against that matching Package.

If a Fallback does occur, then the previous Package needs to be updated to reflect the fact it is now active, and no new Bind is required.

FREE Processing

The Housekeeping Job is the only method by which a FREE PACKAGE is executed.

It is periodically run per DB2 Subsystem (perhaps from the scheduler) and will execute a FREE PACKAGE for any Package meeting the specified criteria.

Packages with a STATUS value of **I** or **D** may be freed. The criteria by which a Package is FREEd is determined according to the parameters listed in Table 3-15.

Table 3-15. FREE Processing

Input Parm	Description
Inactive Keep Days	If the status of a Package is **I** and the status timestamp indicates it has been in this status for at least the same number of days as the specified input Parameter, then the Package is freed.
Delete Keep Days	If the status of a Package is **D** and the status timestamp indicates it has been in this status for at least the same number of days as the specified input Parameter, then the Package is freed.

Testing the Setup

Correct Installation of the Package Versioning functionality should be thoroughly tested to make sure that after it is implemented no Packages are freed when they should not be.

Build a Test Plan

A Test plan should be created to test the various scenarios within the organization. This should include all of the ISPW operations which would affect a Task and its generated DBRM/LOAD.

Compare Against DB2 Catalog

The ISPW DB2 Package Version Table needs to accurately represent what Packages are in the DB2 Catalog. It is recommended that during the testing process comparisons be made between the two.

Ensure Housekeeping Frees Correctly

A key part of the testing is to ensure that the Housekeeping process will not FREE any Packages that it should not.

Chapter 4.
DB2 Generated DECLARE Statements

DCLGENs are source modules generated from the DB2 catalog for a DB2 table or view. DCLGEN processing differs from the typical Generated Type processing because it is the source itself that is generated directly into the primary source dataset.

Environment Assumptions

- By default, the same implementation name/rule specified for an application level that is controlling plan/package processing, is also used for DCLGEN processing to retrieve DB2 control information from the M.AD Plan Implementation entries. Information used includes:

  a. DB2 subsystem
  b. Owner/qualifier value – used to set CURRENT SQLID value to qualify the unqualified table/view names. If for a TEST level no owner or qualifier values are specified, the qualifier used will default to the submitter's userID.

  If a different implementation name/rule applies, then it must be passed to routine WZTDCLG# specifically using parameter IMPL().

- The userID submitting a Generate job must have authority to update the source dataset (either TEST or HOLD) directly as the DCLGEN is generated directly into the source dataset member. The ISPW program normally used for updating HOLD source datasets is not employed here.

Production Cycle Processing

Promote

A pre-exit is executed to empty the source module before it is promoted to the HOLD level. This prevents program generates at this HOLD level from using the DCLGEN before the Generate for the DCLGEN has executed. Otherwise, the DCLGEN may contain declarations that do no properly reflect the table/view structure at this level.

Initialization

M.CT Values

The values in Table 4-1 should be used for the M.CT entry.

Table 4-1. M.CT Values for DCLGENs

Field	Value
Description	DB2 DCLGEN Modules
Cycle Flag	**Y**
Generate Flag	**Y**
Generate Skeleton	Skeleton name for component type

Table 4-1. M.CT Values for DCLGENs (Continued)

Field	Value
Generate Job	Name of the batch job to perform the demanded generate
Generate Table	Name of the table to be used in the demanded job
Test Gen. Panel	Test panel name for component type
Hold Gen. Panel	Hold panel name for component type
Prod Move Job	Currently not used
Model	**DUMMY** (not applicable to DCLGENs)

M.ER Values

For each DB2 subsystem, an M.ER entry is required to describe the DB2 runtime library. As shown in Figure 4-1, the Type Code must be of the format: `DB2R<subsystemid>`.

This runtime library holds the program used to set the CURRENT SQLID value for the Generate job execution. Default name values for the program (DSNTIAD) and plan (DSNTIAD) are set in routine WZTDCLG#. Site names may differ. If so, these values may be overridden.

Figure 4-1. M.ER — Browse External Reference Detail (PROD)

```
ISPW M.ER          BROWSE EXTERNAL REFERENCE DETAIL (PROD)
Command ===>


  Type Code (KEY) ==> DB2DB2T            View Code (KEY) ==>

  Description     ==> DB2 LOAD LIBRARY FOR SUBSYSTEM: DB2T

  Variable/Dsname ==> SYS1.DB2T.DB2.RUNLIB.LOAD
  (Fully qualified if data set name, no quotes)

Press END to return
```

M.EX Values

The values in Table 4-2 should be used for the M.EX entry.

Table 4-2. M.EX Values for DCLGENs

Op	Field	Value
P	Exit Key	The component type
	Description	"Erase DCLGEN source before copy to next level"
	Pre-op	ISPEXEC SELECT CMD(%WZTDCLR# &DEBUG)

Site-specific Components

Sample Components

Each site must customize generate panels and skeletons according to their requirements and guidelines. For reference purposes, sample DCLGEN components are supplied in the

ISPW installation SAMPLIB dataset that may be reviewed to aid in the setup of DB2 DCLGEN processing. These include the samples listed in Table 4-3.

Table 4-3. SAMPLIB Components for Setup of DCLGEN Processing

Type	Name	Description
Panel	WZTDCLGD	TEST level generate example
Panel	WZTDCLGH	HOLD level generate example
Skeleton	WZTDCLG#	Mainline DCLGEN generate skeleton
Skeleton	WZU@JOB	Jobcard
Skeleton	WZU@INPR	Reset task “in process” status

Jobcard Changes for DCLGENs

If DB2 jobs must run under special job classes, be sure the analysts know to specify this class on their jobcard when requesting DB2 services. If a site generates jobcards for the analysts, check for the DCLGEN Component Types to set the proper job class values.

Chapter 5.
Telon Interface

In this discussion, we are assuming that you have some basic knowledge about Telon.

Telon Interface Overview

Processes are provided to support the TDF environment for ISPW test levels using basic Telon processing including invoking the TDF environment to edit, exporting source from TDF into PDS members and importing ESF source from PDS members into TDF. After a Telon task Type has been promoted into a hold level, the exported ESF PDS source is managed in ISPW like any other PDS-based technology Type.

Telon Interface Data Repository

An additional ISPF table is used to store the information necessary to process Telon Types. This table contains 2 categories of Telon information:

- System level information (needed to run Telon itself)
- Application level information (needed for TDF processing)

Default Telon system information is defined for a site with the ability to override these values by application. This enables the support of multiple versions of Telon. Each application using Telon types requires an entry to define its TDF environment. See the Section on Maintenance Options, M.TL later in this chapter for detailed information.

Production Cycle Processing

Checkout

The ESF source, which is stored in the production PDS, is copied to TEST in normal ISPW processing. In the post-exit, the Telon import routine is invoked to import the (non-empty) ESF source module into TDF.

Delete

If defined, delete(s) from the TDF will be processed when a task is removed from the TEST level. If TDFs are kept fully populated, then do not define the ‘D’ exit.

Select

In the pre-exit, the TDF environment is invoked for the application. Normal ISPF edit is bypassed.

Select to Edit in SPF Editor (ST)

In the pre-exit, an export from TDF is performed. The SPF editor is invoked for the PDS member. In the post-exit, the updated PDS module is imported back into TDF.

Promote

In the pre-exit, if the task is in the TEST level, the Telon export routine is invoked to export the ESF source from TDF into the TEST PDS before normal promote processing copies the source to the HOLD PDS.

Regress

In regular processing, the PDS source is moved to the previous level and depending on the setting of the Retain flag for the libraries (M.AD, Names), load and GMT modules are deleted.

In the post-exit, if the task was regressed to a TEST level, the Telon import routine is invoked to import the ESF source into TDF.

Although the Telon export and import routines are invoked automatically in the processing of standard ISPW line command functions, they may also be invoked explicitly by the line commands 'EP' and 'IP'.

Initialization

M.CT Values

An ISPW Component Type is defined for each Telon source type to be managed. Example values are shown in Table 5-1.

Note: If target environment can be more than one value for a given Component Type, contact Compuware Customer Support to discuss alternatives for setup.

Table 5-1. M.CT Values for Telon Source Types

Field	Value
Description	**TELON Modules**
Technology	**TL**
ISPW Type	The two-character Telon source type. Examples: • BD • DR • SD
Cycle Flag	**Y**
Generate Flag	**Y**
Generate Skeleton	Skeleton name for component type
Generate Job	Name of the batch job to perform the demanded generate
Generate Table	Name of the table to be used in the demanded job
Test Gen. Panel	Test panel name for component type
Hold Gen. Panel	Hold panel name for component type
Prod Move Job	Currently not used
Execution Env.	The one-character Telon target indicator. Examples: • C – CICS • B – Batch • I – IMS

M.EX Values

The values listed in Table 5-2 should be used for the M.EX entries.

Table 5-2. M.EX Entry Values for Telon

Op	Field	Value
C	Exit Key	TL
	Description	**ISPW Telon import into TDF VSAM after checkout**
	Post-op	ISPEXEC SELECT CMD(%WZTTL#IP &DEBUG)
D	Exit Key	TL
	Description	**ISPW Telon delete from TDF VSAM post-op**
	Post-op	ISPEXEC SELECT CMD(%WZTTLD# &DEBUG)
EP	Exit Key	TL
	Description	**ISPW Telon export from TDF VSAM into PDS**
	Pre-op	ISPEXEC SELECT CMD(%WZTTL#EP &DEBUG)
IP	Exit Key	TL
	Description	**ISPW Telon import into TDF VSAM from PDS**
	Pre-op	ISPEXEC SELECT CMD(%WZTTL#IP &DEBUG)
R	Exit Key	TL
	Description	**ISPW Telon export from TDF VSAM before promote**
	Pre-op	ISPEXEC SELECT CMD(%WZTTL#EP &DEBUG)
S	Exit Key	TL
	Description	**ISPW Telon TDF VSAM edit**
	Pre-op	ISPEXEC SELECT CMD(%WZTTLS# &DEBUG)
ST	Exit Key	TL
	Description	**ISPW Telon export before SPF edit; Import after**
	Pre-op	ISPEXEC SELECT CMD(%WZTTL#EP &DEBUG)
	Post-op	ISPEXEC SELECT CMD(%WZTTL#IP &DEBUG)
X	Exit Key	TL
	Description	**ISPW Telon import into TDF VSAM after regress**
	Post-op	ISPEXEC SELECT CMD(%WZTTL#IP &DEBUG)

Example M.AD Associations

The libraries defined for an application in M.AD for a Telon source type will be PDS datasets for all levels in which the Telon ESF is stored. The associations defined for an application in M.AD for a Telon Driver Type (for example, TLDR) might be defined as shown in Figure 5-1.

Figure 5-1. M.AD — BLM Associations Table (PROD)

```
WZZMADA D/A       BLM  ASSOCIATIONS TABLE (PROD)                  UPDATE MODE
Command  ===>                                                  Scroll ===> CSR

List Commands: A Add Entry, L Locate Entry, B Browse Mode
Line Commands: D Delete

                                 Target                Assoc  Assoc  Assoc
    Type   Clas   Association    Env.   Set  Seq.      Appl   Type   Clas

    TLDR               C               LKED  1       BLM    LKED
    TLDR               G               GMT1  1       BLM    DBRM
    TLDR               G               LOAD          BLM    LODG
    TLDR               I               DCL   1       BLM    DCLG
    TLDR               I               MAC   1       BLM    COPY
    TLDR               I               SYS   1       BLM    LODG
```

Site-specific Components

Sample Components

Each site must build very customized generate panels and skeletons according to their requirements and guidelines. For reference purposes, sample and model components supplied in the ISPW installation SAMPLIB dataset may be reviewed to aid in the setup of Telon processing. These include the samples listed in Table 5-3.

Table 5-3. SAMPLIB Components for Setup of Telon Processing

Type	Name	Description
Panel	WZUGD	TEST level generate example
Panel	WZUGH	HOLD level generate example
Skeleton	WZUG	Generalized mainline “driver” skeleton
Skeleton	WZU@JOB	Jobcard
Skeleton	WZU@MTYP	Sets parameters by Component Type
Skeleton	WZU@TLN	Telon generate pre-compile step
Skeleton	WZU@TLN2	Telon post step to process format produced
Skeleton	WZU@DB2	DB2 pre-compile step
Skeleton	WZU@COB	COBOL compile step
Skeleton	WZU@LKED	Link-edit step
Skeleton	WZU@DBRM	Copy of temp DBRM to PDS
Skeleton	WZU@DB2B	DB2 bind processing
Skeleton	WZU@INPR	Reset task “in process” status

Export Processing

A Telon export is performed when promoting from a TEST level to HOLD in the foreground to capture the ESF source and “freeze” it. This is necessary to maintain source-to-load synchronization at the HOLD levels while allowing a task to be re-added to TEST for further modification.

For generates performed at the TEST level, the export processing is coded in the skeletons, not performed in the foreground. This ensures getting the current source and moves the machine cycles required for the export to the background.

Maintenance Option M.TL

ISPW's maintenance option M.TL defines the parameters used in the Telon interface.

Main M Option Menu

Option TL can be added to the main maintenance option screen in the section for Special Technologies. An example is shown in Figure 5-2.

Figure 5-2. ISPW Administrator Maintenance Screen

```
ISPW M          ISPW ADMINISTRATOR MAINTENANCE (PROD)
COMMAND ===>


  Reference Data ( R )            Support

  AP  Approval Rules              CH  Change Codes
  AD  Application Definitions     CR  Component Reference
  CT  Component Types             LM  Library Management
  ER  External References         P   Prefix Numbers
  EX  Exits                       SC  Set Class Definitions
                                  SM  Server Maintenance
                                  U   User List

  General                        Special Technologies

  C   Comment File               TL  Telon
  CO  Commands "O"               T2  Technology 2
  CZ  Commands "Z"               T3  Technology 3
  Z   Generate Parms Tables

Enter END command to return to the previous menu.
```

Main Telon Screen

Figure 5-3 shows the main Telon interface screen.

Figure 5-3. Main Telon Screen

```
ISPW M.TL -------------- TELON INTERFACE TABLE (PROD) ------------ Browse only
COMMAND ===>                                                SCROLL ===> CSR

  Commands: L(ocate)               Line Commands: S

  APPLID LEVEL
   ----  ----
   $$$$  $$$$
   BLM   $$$$
   BLM   FIXT
   BLM   TEST
   SITE  TEST
****************************** Bottom of data ********************************
```

Telon Table

The ISPF table $TELLIB is used to store the information necessary to run Telon. This includes:

- the Telon system library names,
- application TDF library names, and
- the DB2 subsystem ID and plan name to be used if invoking Telon TDF under DB2.

Special Notes

The Telon interface table contains two categories of information:

System level information (level=$$$$)

A key value of `level=$$$$` indicates Telon system level information. The site default values are stored in key:

- `APPLID=$$$$`
- `LEVEL=$$$$`

In order to support the use of multiple versions of Telon, the site system defaults may be overridden for an application by building the entry key:

- `APPLID=<appl>`
- `LEVEL=$$$$`

Application level TDF information (TEST levels only)

The Telon information needed to define an application's TDF environment is stored in the key:

- `APPLID=<appl>`
- `LEVEL=<TEST level>`

Note: The TDF environment is supported for TEST levels only.

Field Descriptions

Table 5-4 describes each field for an external reference definition.

Table 5-4. Telon External Reference Definition Fields

Field Name	Description
Application (key)	Use **$$$$** to specify the site-wide default system information. Otherwise, specify a valid application code. Valid values: • $$$$ • a valid application code.
Level (key)	Use **$$$$** to specify system information. Otherwise, level should be a valid TEST level to specify the application's TDF environment. Valid values: • $$$$ • a valid TEST level name.
System and TDF library name fields	Enter the values for the parameters as indicated.
DB2 SSID	If TDF is invoked under DB2, enter the DB2 subsystem ID to be used. Valid values: • A valid DB2 subsystem ID. • NONE – DB2 is not used. • Blank – when defining application level TDF information, the value specified on the system "$$$$" entry is used.
DB2 Plan Name	If TDF is invoked under DB2, enter the plan name to be used. Valid values: • Plan name. • Blank – when defining application level TDF information, the value specified on the system "$$$$" entry is used.

Detail Screen – Site Info

Figure 5-4 shows the definition of site-wide default system information.

Figure 5-4. Telon Interface System Information - Default System Information

```
ISPW M.TL ------------- TELON INTERFACE SYSTEM INFORMATION (PROD) ------------
COMMAND ===>


Applid (KEY) ==> $$$$
Level  (KEY) ==> $$$$

TELON SYSTEM INFORMATION:

User Prefix      ==> &SYSUID..TELON
Telon Hold       ==> SYS3.TELON.TNHOLD
Telon Help       ==> SYS3.TELON.TNHELP
Telon Load       ==> SYS3.TELON.LOAD
Telon Load 2     ==> SYS3.TELON.PT3LOAD
PTPTCH CNTL      ==> SYS3.TELON.INSTALL(CINPTPCH)
Maclib           ==> SYS3.TELON.MACLIB
CMACLIB          ==> SYS3.TELON.MACLIB
COBOL Copylib    ==> SYS3.TELON.SOURCEC

    DB2 SSID ==> NONE DB2 Plan ===> XXXXXXXX


PRESS ENTER TO CONTINUE, END TO TERMINATE
```

Detail Screen – Application Info

Figure 5-5 and Figure 5-6 show the definition of application (override) system information.

Figure 5-5. Telon - Defining Application (Override) System Information, Part 1

```
ISPW M.TL ------------- TELON INTERFACE SYSTEM INFORMATION (PROD) ------------
COMMAND ===>


Applid (KEY) ==> BLM
Level  (KEY) ==> $$$$

TELON SYSTEM INFORMATION:

User Prefix      ==> &SYSUID..TELON
Telon Hold       ==> SYS3.TELON.V23.TNHOLD
Telon Help       ==> SYS3.TELON.V23.TNHELP
Telon Load       ==> SYS3.TELON.V23.LOAD
Telon Load 2     ==>
PTPTCH CNTL      ==> SYS3.TELON.V23.INSTALL(CINPTPCH)
Maclib           ==> SYS3.TELON.V23.MACLIB
CMACLIB          ==> SYS3.TELON.V23.MACLIB
COBOL Copylib    ==> SYS3.TELON.V23.SOURCEC

    DB2 SSID ==> DB2T DB2 Plan ===> BLMTDF


PRESS ENTER TO CONTINUE, END TO TERMINATE
```

Figure 5-6. Telon - Defining Application (Override) System Information, Part 2

```
ISPW M.TL ----------- TELON INTERFACE APPLICATION INFORMATION (PROD) ---------
COMMAND ===>


Applid (KEY) ==> BLM
Level (KEY)  ==> TEST

TELON APPLICATION INFORMATION:

CCL Custom Code      ==> BLM.TELON.TNTCCL
DD  Data Admin       ==> BLM.TELON.TNTDD
DDW Data Admin Work  ==> BLM.TELON.TNTDDW
DF  Entity           ==> BLM.TELON.TNTDF
DFW Entity Work      ==> BLM.TELON.TNTDFW
DX  DB Admin         ==> BLM.TELON.TNTDX
DXW DB Admin Work    ==> BLM.TELON.TNTDXW

    DB2 SSID ==> DB2T DB2 Plan ===> TDFPLANC


PRESS ENTER TO CONTINUE, END TO TERMINATE
```

Chapter 6.
QMF Interface

In this discussion, we are assuming that you have some basic knowledge about QMF.

QMF Interface Overview

Forms, Queries, and Procs are the only QMF objects managed by ISPW. Datasets (PDS/PDSE) are used by ISPW to maintain the QMF object source. The source is imported from these datasets into the appropriate QMF Environment and under the correct Owner as required. All the necessary Exits are provided to manage these QMF objects via standard ISPW Processing.

How ISPW Interacts with QMF

ISPW makes use of the "REXX Callable Interface". This allows for QMF commands to be invoked from within a REXX exec. When ISPW first needs to communicate with QMF (for example, when selecting a task for edit), it will issue a call to QMF to initiate communication. QMF is then effectively running alongside ISPW, and ISPW can issue QMF commands (for example, EXPORT, SAVE, etc.) for immediate action.

QMF remains active until the user exits from the Assignment they are working in. Note that during the time that QMF is active, QMF cannot be started again in the user's TSO session (for example, in a split screen). This is a QMF constraint.

QMF tasks can be freely added to any assignment with other non-QMF tasks.

Environment Assumptions

- QMF version 2.2 or higher.
- Each module is saved in QMF with CONFIRM=no and SHARE=yes parameters.
- The module is saved under the same name at each level, and this name is the Assignment Task name.
- At the TEST level, the owner is the same DB2 Owner as defined in the M.AD options for that application. The QMF interface will issue a SET CURRENT SQLID = db2owner at the TEST level.
- A batch job running under a production userID is used to save the modules to QMF in both the HOLD and PROD environments. This userID needs authority within DB2 to save the objects under the owner defined in M.AD.
- Modules within QMF are not erased as they are promoted up the levels.

QMF Object Ownership

QMF objects have to be saved under an Owner ID in QMF. The Owner is specified in the Reference Data under M.AD (Plan Implementations). An entry with an Implementation of "QMF" can be defined in M.AD if the Owner of the QMF objects will be different from the Owner of the DB2 Plans and Packages.

It is normal that the ISPW User does not have any authority to save QMF objects under the Owners specified in the HOLD and PROD Environments (for example, all ISPW levels above the TEST level). The QMF Interface provides a mechanism where either a batch job is submitted under a controlled userID to perform this operation or the Set Processor does this under its userID.

Working at the TEST Level

The QMF Interface provides two ways of working at the TEST level:

- The user has authority to save QMF Objects under the Owner as specified in the Reference data. This is the default way of working. While the user is working on QMF objects within ISPW, a "SET CURRENT SQLID = testowner" is issued so that the objects are being saved under the relevant Owner.
- The user operates in QMF under their own userID, and the save is done under a controlled userID by the submission of a generate job. If the flag settings in M.AD specify for a TEST Generate to be done, the QMF interface will assume that the User will be working in QMF under their own ID.

Production Cycle Processing

C – Checkout

If this is a new module and a model exists, or if this module already exists in ISPW, the module (or model) is copied to the TEST PDS, imported into the "TEST" QMF and saved. If this is a new module and no model exists, a TEST PDS member is created, but nothing is saved in QMF.

B – Browse

For Procs and Queries, ISPW will use standard PDS processing to browse the members in the PDS. Form information is difficult to understand and review in a PDS, so it is displayed in QMF and the User placed in the QMF Form manager. Regardless of the status of the module (that is, whether it is in HOLD, PROD, etc.), it will always be reviewed in the TEST QMF Form manager because TEST QMF is already active. This may involve an import if the Form is not already in the TEST environment, but it is not saved on exit.

D – Delete

The Task is deleted along with the copy in the PDS. The user is given the option of having it deleted from the QMF TEST environment.

S – Select (Default Editor)

The default editor for QMF objects can either be QMF itself or the ISPF editor. This is can be specified for each of the QMF types by defining entries in M.ER to indicate this.

Whichever editor is used, the source will be saved in both the source dataset and QMF itself.

Note: When editing in QMF, whatever is currently in the QMF temporary storage area is what QMF saves. If you had selected a Proc and then edited another Proc before returning to the Assignment, the second Proc will be saved into the module in the Assignment. The same applies to Queries and Forms.

ST – Select (Alternate Editor)

This operation will invoke either QMF or ISPF Edit, depending upon the entries in M.ER that specify the default editor. (This will invoke the non-default editor.)

Whichever editor is used, the source will be saved in both the source dataset and QMF itself.

Note: When editing in QMF, whatever is currently in the QMF temporary storage area is what QMF saves. If you had selected a Proc and then edited another Proc before returning to the Assignment, the second Proc will be saved into the module in the Assignment. The same applies to Queries and Forms.

P – Promote

The job that does the Promote must have authority to save the QMF object under the owner specified for the level to which the promotion is taking place.

Two methods of promotion are possible and depend upon the Flag settings in M.AD:

- Promote from within the User address space and demand a controlled job to do the import into the QMF Environment, or
- Create a Set to do the promote, in which case the Set Processor will perform the QMF Import under its own authority.

For each promotion level in the change cycle, a decision needs to be made as to which method of promotion is to be used. It is important that the flag settings in M.AD and Approval Rules in M.AR are set up correctly for the chosen method at each level. These settings are discussed in more detail later in this section.

Modules are ***not*** erased from the QMF Environment being promoted from.

RE – Rename

The object in QMF is also renamed if the user has the authority to do so.

EP – Export

If the module exists in QMF, it is exported to the TEST PDS.

When one module is selected for update, it is very convenient to make changes to related modules without returning to the Task List. Returning from QMF will only save the original module selected. The source in the TEST PDS is out of sync with QMF for all the other updated modules. To save these other updated modules, this option can be used to move the QMF source back into the PDS member.

IP – Import

The option imports the TEST PDS member into QMF and saves it.

This would be useful if the TEST PDS member was edited directly out of ISPW.

EX – Execute

Only Queries and Procs can be executed. ISPW will simply invoke an execution of the proc or query. QMF will prompt for any parameters before execution.

Action "D" Processing

As the Task is promoted through the levels, the QMF object will be erased from the QMF Environments.

Reference Data Entries

M.CT Values

An ISPW Component Type is defined for each of the three QMF Object types. Sites may choose their own Component Type code for each of these, however the other attributes should be as shown in Table 6-1.

Table 6-1. QMF Object Type Attributes

Field	Value
Description	**QMF Queries** or **QMF Procs** or **QMF Forms**
Technology	**QM**
ISPW Type	QQRY for Queries; QPRC for Procs; QFRM for Forms
Cycle Flag	**Y**
Generate Flag	**Y**
Generate Skeleton	WZUQMG# (from the SAMPLIB dataset), modified for your site
Generate Job	Name of the batch jobname to be used to perform the demanded generate
Generate Table	Name of the table to be used in the demanded job
Test Gen. Panel	WZUQMGD (from the SAMPLIB dataset) **Note:** This panel only needs to be defined if users need to submit a job at the TEST level to save the QMF Object under the Test level Owner.
Hold Gen. Panel	WZUQMGH (from the SAMPLIB dataset) **Note:** This panel only needs to be defined if the settings in M.AD specify that a demanded job is required at the Hold Level to save the QMF Object under the Hold level Owner. Alternative processing can be defined where the Set Processor does this for all Hold Levels.
Prod Move Job	Currently not used
Execution Env.	spaces

M.EX Values

The values listed in Table 6-2 should be used for the M.EX entries.

Table 6-2. M.EX Values for QMF

Op	Field	Value
B	Exit Key	QM
	Description	Browse Form in QMF
	Pre-op	ISPEXEC SELECT CMD(%WZTQMB# &DEBUG)
C	Exit Key	QM
	Description	Import QMF Object into QMF Environment
	Post-op	ISPEXEC SELECT CMD(%WZTQMC# &DEBUG)

Table 6-2. M.EX Values for QMF (Continued)

Op	Field	Value
D	Exit Key	QM
	Description	Delete QMF Object from QMF
	Post-op	ISPEXEC SELECT CMD(%WZTQMD# &DEBUG)
EP	Exit Key	QM
	Description	Export QMF Object to the source dataset
	Pre-op	ISPEXEC SELECT CMD(%WZTQMEP &DEBUG)
EX	Exit Key	QM
	Description	Execute QMF Object
	Pre-op	ISPEXEC SELECT CMD(%WZTQMEX &DEBUG)
G	Exit Key	QM
	Description	Set QMF variables for Import
	Pre-op	ISPEXEC SELECT CMD(%WZTQMG# &DEBUG)
IP	Exit Key	QM
	Description	Import QMF Object from source dataset
	Pre-op	ISPEXEC SELECT CMD(%WZTQMIP &DEBUG)
P	Exit Key	QM
	Description	Import QMF Object to next level
	Pre-op	ISPEXEC SELECT CMD(%WZTQMP# &DEBUG)
S	Exit Key	QM
	Description	Invoke default editor for QMF
	Pre-op	ISPEXEC SELECT CMD(%WZTQMS# &DEBUG)
ST	Exit Key	QM
	Description	Invoke alternate editor for QMF
	Pre-op	ISPEXEC SELECT CMD(%WZTQMST &DEBUG)
X	Exit Key	QM
	Description	Import into previous QMF environment
	Post-op	ISPEXEC SELECT CMD(%WZTQMX# &DEBUG)

M.ER Values

The values listed in Table 6-3 should be used for the M.ER entries.

Table 6-3. M.ER Values for QMF

Key	Value
QMQUAL	This should contain the name of the Owner of the “SET CURRENT SQLID” Query.
QMEDITQ	If this entry is defined and has a value of “QMF”, the default editor invoked for QMF Queries with the ISPW “S” Operation will be QMF itself. Otherwise, the default editor will be ISPF Edit. ISPW Operation “ST” will invoke the non-default editor.
QMEDITF	Same as above, except for QMF Forms.
QMEDITP	Same as above, except for QMF Procs.

Example M.AD Flags

The M.AD (Flags) settings are crucial to the way that the QMF Interface works. Figure 6-1 shows an example of Flag settings in M.AD.

Figure 6-1. Sample M.AD Flag Settings - QMF

```
ISPW M.AD/F       APPLICATION QMF STREAM QMF FLAGS                UPDATE MODE
Command  ===>                                                  Scroll ===> CSR

List Commands: L Locate Entry, B Browse Mode

                  Promote  Version   Generate   Implement   Pack  Keep
Type Clas  Lev    Method   Control   Opt  Chk   Opt  Chk    Src   Memb

QFRM       CONS
QFRM       HOLD     A                R    Y
QFRM       PROD
QFRM       TEST                      R    Y
QPRC       CONS
QPRC       HOLD     A                R    Y
QPRC       PROD
QPRC       TEST                      R    Y
QSQL       CONS
QSQL       HOLD     A                R    Y
QSQL       PROD
QSQL       TEST                      R    Y
******************************** Bottom of data ********************************
```

The above Flag settings would be specified for the following situation:

- Level Structure is TEST-CONS-HOLD-PROD
- The CONS level is a simple staging area and has no QMF Environment. The corresponding M.AD (Plan Imp) entry would not specify a DB2 Owner for the CONS level.
- Users have no authority to save the QMF Objects under the TEST Owner, which is why the Generate is required at TEST. (This does the import under the demanded job ID.)
- Promoting from CONS to HOLD can be done either in a Set or online using the “P” Operation. Either way, a job will be submitted to do the import into the QMF Environment.
- Promoting to PROD can only be done in a Set, so Approval Rules (M.AR) should also be specified to force this to happen.
- The Generates that do the imports must be successfully completed before promotion to the next level (Generate Chk set to “Y”).

M.AD Plan Implementation Entries

The QMF Interface uses the values specified in M.AD (Plans) to determine the DB2 subsystem and QMF Object Owners for the levels in ISPW. If these values are different from the values used for DB2 Plan and Package bindings, QMF-specific entries can be defined using an Implementation of “QMF”.

Figure 6-2 shows the list of entries in M.AD (Plans) for an ISPW Application and Stream. Separate entries have been defined specifically for QMF definitions.

Figure 6-2. Entries in M.AD for Application and Stream

```
ISPW M.AD/T       APPLICATION QMF STREAM QMF PLANS                Row 1 of 6
Command  ===>                                                  Scroll ===> CSR

List Commands: A Add Entry, L Locate Entry, B Browse Mode
Line Commands: S Select, D Delete

  Level  Implementation

  HOLD
  HOLD   QMF
  PROD
  PROD   QMF
  TEST
  TEST   QMF
****************************** Bottom of data ********************************
```

Figure 6-3 shows how the DB2 subsystem and QMF Owner are specified. The DB2 Plan/Package fields are superfluous.

Figure 6-3. Specifying DB2 Subsystem and QMF Owner

```
ISPW M.AD/P        MODIFY APPLICATION QMF STREAM QMF PLAN IMPLEMENTATIO
Command ===>

Enter required details:

 Level  (KEY) ==> HOLD
 Impl   (KEY) ==> QMF      (Implementation name)
 Do BINDs?    ==>    (Y/N) N=No bind processing for PLANs or PACKAGEs
 Owner        ==> QMFHOLD          Qualifier    ==>
 Subsystem    ==> DSN1             Explain      ==>           (Y/N)

 Logical  Collection IDs ===>
 Physical Collection IDs ===>

Plan Bind Details :
 Name Prefix  ==>         Name Suffix  ==>
 Other Parms  ==>

Package Bind Details :
 Collection   ==>
 Other Parms  ==>

Press ENTER to complete the change or END to terminate
Note: You can add a new entry by overtyping the Keys with new unique values
```

Site-specific Components and Install Notes

Sample Components

A certain amount of customization is required for the QMF Interface to be installed properly. For reference purposes, sample and model components supplied in the ISPW installation SAMPLIB dataset may be reviewed to aid in the setup of QMF processing. These include the samples listed in Table 6-4.

Table 6-4. SAMPLIB Components for Setup of QMF Processing

Type	Name	Description
Panel	WZUQMGD	TEST level generate example
Panel	WZUQMGH	HOLD level generate example
Skeleton	WZUQMG#	Generalized mainline “driver” skeleton

Table 6-4. SAMPLIB Components for Setup of QMF Processing (Continued)

Type	Name	Description
REXX	WZUQMAL	Allocate QMF datasets and start QMF
QMF Query	WZTQM#U	Query to be imported into QMF which does "SET CURRENT SQLID"

QMF Dataset Allocation and Startup

The WZUQMAL CLIST is used to invoke QMF alongside ISPW. This allocates the QMF environment for the user when the first QMF type module is accessed (where the action requires QMF) within an Assignment. If you already have a QMF start-up CLIST, you can modify WZUQMAL to call it instead. The CLIST has sections in it for allocating the QMF datasets, both for foreground processing and background processes.

Copy the WZUQMAL CLISTs to your site ISPW CLIST dataset. Modify it as described above to initiate the QMF environment for use within ISPW.

Query for "SET CURRENT SQLID" Processing

A specific QMF Query accessible by all (SHARE=YES) is required to perform the "SET CURRENT SQLID =" processing. This should exist in all DB2 subsystems under the ownership of the userID specified in the M.ER entry for "QMQUAL".

Default Editor Settings

M.ER entries for QMEDITQ, QMEDITF, and QMEDITP need to be defined with a value of "QMF" if QMF is to be the editor of choice when the "S" operation is performed. It is normal for this to be set for QMF Forms (QMEDITF) because these cannot easily be edited in the ISPF editor. If the default editor for a specific object type is to be the ISPF editor, do not define the M.ER entry corresponding to that type. The "ST" operation will always invoke the alternative editor to what is invoked on an "S" operation.

Requirements for Generate Processing

How the QMF Interface is implemented will determine whether or not Generates are required at the TEST and HOLD levels. The four situations listed in Table 6-5 determine what is required.

Table 6-5. QMF Interface Implementation Situations

Situation	Installation Considerations
Users have no authority to save QMF Objects under their own UserID	A TEST Generate must be set up to perform the QMF Import for the TEST Owner. The skeleton WZUQMG# and panel WZUQMGD need to be copied into the Site libraries and modified as required. The M.AD Flag settings should define for a Generate to be done at TEST.
Users have the authority to save QMF Objects at the TEST level under their own UserID	No Generate is required, so the M.AD Flag settings should not specify for a Generate to be done at TEST.
Promotion to a particular level can be done both Online and in a Set	If Users will be doing "online" promotes, a HOLD Generate must be set up to perform the QMF Import for the Owner at the level being promoted to. The skeleton WZUQMG# and panel WZUQMGH need to be copied into your Site libraries and modified as required. The M.AD Flag settings should define for a Generate to be done at each of the levels for which this is done.

Table 6-5. QMF Interface Implementation Situations (Continued)

Situation	Installation Considerations
Promotion to a particular level can only be done in a Set	This is the normal situation for a promotion to Production, but this can also be the rule for all HOLD levels if required. Be sure to have an Approval Rule specified in M.AR that forces all promotes through the Set Processor. For levels where this is specified, the M.AD Flag setting should indicate that no Generate is required and that the "Promote Method" specified for the level being promoted ***from*** must be set to "A".

Chapter 7.
Natural Interface

In this discussion, we are assuming that you have some basic knowledge about Natural.

Natural Interface Overview

ISPW manages most types of Natural objects including Programs, Maps, Local and Global data Areas, Copy Code, etc. Datasets (PDS/PDSE) are used by ISPW to maintain the Natural source. The source is loaded from these datasets into the appropriate Natural Environment and under the Application Library as required. All the necessary Exits are provided to manage these Natural objects via standard ISPW Processing.

How ISPW Interacts with Natural

ISPW uses the Natural TSO and batch interface load modules to interface with Natural. Each call allocates the required Natural files to access the correct Natural region. When using the Natural TSO interface load module, ISPW passes the commands via the STACK parameter. When using the Natural batch interface load module, the commands are passed through the CMSYNIN DD card.

Most processes initiated against a Natural component will start up Natural, process the prepared commands, and terminate the Natural session. The Edit process does not terminate Natural, but leaves the developer in Natural for editing of other components or testing. The developer will end the Natural session manually. In either case, the termination of the Natural session brings the developer back to the ISPW screen from which the Natural process was initiated.

Natural tasks can be freely added to any assignment with other non-Natural tasks.

Environment Assumptions

- Natural version 2.2.6 or higher.
- The module is saved under the same name at each level, and this name is the Assignment Task name.
- The module is loaded to and unloaded from the Natural library defined in M.AD for the application/level that the module is currently at.
- In the Test environment, all loads and unloads are performed using the developer's userID, and therefore the developer must have the necessary access in Natural.
- A batch job and/or a started task, running under an ISPW userID, is used to load the modules at both the ISPW HOLD and PROD environments. This userID needs authority within Natural to log onto the Natural Library and save the object into that Natural Library.
- Modules within Natural are not normally erased because they are promoted up the levels, but this is managed by the M.AD retain flag.

The Natural Load

Natural objects are loaded into the Natural library specified in the Reference Data under M.AD/NA (Natural). An ISPW application/level definition can point at only one Natural Library. It is normal that the ISPW user does not have any authority to load Natural objects into the Natural environments defined for the ISPW HOLD and PROD Environments (for example, all ISPW levels above the ISPW TEST level). The Natural Interface provides a mechanism where either a batch job is submitted under a controlled userID to perform this operation or the Set Processor does this under its userID.

Working at the TEST Level

Natural objects are loaded and unloaded in the ISPW Test Environment using the authority of the developer's userID. Therefore, the developer's userID must have update authority in the corresponding Natural environment.

Production Cycle Processing

C – Checkout

If this is a new module and a model exists, or if this module already exists in ISPW, the module (or model) is copied to the TEST PDS, loaded into the "TEST" Natural Environment and saved. If this is a new module and no model exists, a TEST PDS member is created, but nothing is saved in Natural.

B – Browse

When Browsing, ISPW starts up the appropriate online Natural environment, does a Logon to the required library, then does a List of the requested module. When done, ISPW ends the Natural session without allowing any other processing while in the Natural Environment. The Natural Environment source and the PDS source modules are not changed. No Load or Unload is done as part of a Browse. When the Natural session is terminated, the ISPW user is returned to the task list screen.

D – Delete

The Task is deleted along with the source module in the PDS and the copy of the source in the Natural Environment at the TEST level. An attempt is made to repair the TEST Natural Environment (if retain is yes) by loading the source from a PDS at a higher level in the Path to Production. If this source module does not exist, the Production source is used. This repair consists of a load into the TEST Natural Environment and Stow of the module.

S – Select

The editor for Natural objects is the Natural editor. ISPW starts up an online Natural session, logs onto the appropriate Natural application Library, and edits the module. When the edit is finished, ISPW returns to the Natural Development screen, leaving the developer in Natural to do other editing or testing that may be required. When the developer terminates the Natural session, ISPW unloads the first object edited and returns to the ISPW screen from which the original Edit was issued.

Other modules can be edited while in the one Natural session, but they should first be added to an assignment and checked out. Any changes made to modules before the checkout is done will be lost during the checkout. If other modules are edited while in the Natural session, they will have to be manually unloaded to the PDS using the EP

(EXPORT) line command. If this is not done, the wrong version will be passed to a checkout in another path if this version is selected.

P – Promote

The job that does the Promote must have authority to save the Natural object into the Natural Library associated with the level to which the promotion is taking place.

Two methods of promotion are possible and depend upon the Flag settings in M.AD:

- Promote from within the User address space and demand a controlled job to do the load into the Natural Environment; or
- Create a Set to do the promote, in which case the Set Processor will perform the Natural load under its own authority.

For each promotion level in the change cycle, a decision needs to be made as to which method of promotion is to be used. It is important to set up the flag settings in M.AD and Approval Rules in M.AR correctly for the chosen method at each level. These settings are discussed in more detail later in this section.

Modules are ***not***, by default, erased from the Natural Environment being promoted from. This is controlled by the M.AD flag, which defines whether or not a module is to be kept at a level upon promotion. Normally this should be set to Y(es) for Natural modules.

When promoting from the ISPW Test Environment, the Natural object is first unloaded from the Test Natural Environment into the Test PDS before the copy from the Test PDS to the Hold PDS. This ensures the last Natural Environment copy is in sync with the PDS version.

RE – Rename

The source and object in the Natural Environment are deleted, and the module is renamed in the PDS. If the module is flagged to be retained at this level, a previous version is loaded and stowed into this Natural Environment. If the rename is back to the original name, the module is loaded back into the Natural Environment and Stowed.

EP – Export

If the module exists in Natural, then it is unloaded to the TEST PDS. This operation is only allowed in the ISPW Test Environment.

When one module is selected for update, it is very convenient to make changes to related modules without returning to the Task List. Returning from Natural will only save the original module selected. The source in the TEST PDS is out of sync with Natural for all the other updated modules. To resynchronize these other updated modules, this option can be used to unload the Natural source back into the PDS member.

IP – Import

This option loads the TEST PDS member into Natural. This operation is only allowed in the ISPW Test Environment.

X – Regress

This option performs the following steps:

1. Copies the current level PDS version to the previous level PDS version.
2. Deletes the current level PDS version.

3. If the retain flag is on for the current level, ISPW demands the gen process—which uses the next higher level PDS version found—load and Stow at the current level, thus repairing the level.

Note: When the Natural object is copied to the previous level, the load is ***not*** performed at the previous level. This must be done with the EP in Test, or G in the Hold Environments.

Action "D" Processing

As the Task is promoted through the levels, the Natural object will be erased from the Natural Environments.

Reference Data Entries

M.CT Values

An ISPW Component Type is defined for each of the three Natural Object types. Sites may choose their own Component Type code for each of these, however the other attributes should be as shown in Table 7-1.

Table 7-1. Natural Object Type Attributes

Field	Value
Description	See the ISPW Type field below for examples
Technology	**NA**
ISPW Type	**NATC** – Natural Copybooks **NATG** – Natural Global Data Areas **NATH** – Natural Help screens **NATL** – Natural Local Data Areas **NATM** – Natural Sub-programs **NATP** – Natural Programs **NATS** – Natural Subroutines
Cycle Flag	**Y**
Generate Flag	**Y**
Generate Skeleton	WZUG (from the SAMPLIB dataset), modified for your site
Generate Job	Name of the batch jobname to be used to perform the demanded generate
Generate Table	Name of the table to be used in the demanded job
Test Gen. Panel	WZUGD (from the SAMPLIB dataset)
Hold Gen. Panel	WZUGH
Prod Move Job	Currently not used
Execution Env.	spaces

M.EX Values

The values in Table 7-2 should be used for the M.EX entries.

Table 7-2. M.EX Values for Natural

Op	Field	Value
B	Exit Key	NA
	Description	Natural Browse
	Pre-op	ISPEXEC SELECT CMD(%WZTNAB# &DEBUG)
C	Exit Key	NA
	Description	Natural Post Check-out Load
	Post-op	ISPEXEC SELECT CMD(%WZTNAC# &DEBUG)
D	Exit Key	NA
	Description	ISPW Natural Delete
	Pre-op	ISPEXEC SELECT CMD(%WZTNAD# &DEBUG)
EP	Exit Key	NA
	Description	Natural Unload
	Pre-op	ISPEXEC SELECT CMD(%WZTNAEP &DEBUG)
IP	Exit Key	NA
	Description	Natural Load
	Pre-op	ISPEXEC SELECT CMD(%WZTNAIP &DEBUG)
P	Exit Key	NA
	Description	Natural Post Promote Import and Gen
	Post-op	ISPEXEC SELECT CMD(%WZTNAP# &DEBUG)
R	Exit Key	NA
	Description	Natural R exits
	Pre-op	ISPEXEC SELECT CMD(%WZTNAR# &DEBUG)
	Post-op	ISPEXEC SELECT CMD(%WZTNAP# &DEBUG)
S	Exit Key	NA
	Description	Natural Edit and Post Edit Unload
	Pre-op	ISPEXEC SELECT CMD(%WZTNAS# &DEBUG)
	Post-op	ISPEXEC SELECT CMD(%WZTNAEP &DEBUG)
X	Exit Key	NA
	Description	Natural Exit to Repair Current Level
	Post-op	ISPEXEC SELECT CMD(%WZTNAX# &DEBUG)

Example M.AD Flags

The M.AD Flag settings are crucial to the way that the Natural Interface works. Figure 7-3 shows an example of Flag settings in M.AD.

Figure 7-1. Sample M.AD Flag Settings - Natural

```
ISPW M.AD/F       APPLICATION PLAY STREAM NAT FLAGS                UPDATE MODE
Command  ===>                                                  Scroll ===> CSR

List Commands: L Locate Entry, B Browse Mode

                  Promote  Version   Generate   Implement   Pack  Keep
Type Clas  Lev    Method   Control   Opt  Chk   Opt  Chk    Src   Memb

NATM       CONS     A                 R                           Y
NATM       HOLD     A                                             Y
NATM       PROD
NATM       TEST     R                 R
NATP       CONS     A                 R                           Y
NATP       HOLD     A                                             Y
NATP       PROD
NATP       TEST     R                 R
NATS       CONS     A                 R                           Y
NATS       HOLD     A                                             Y
NATS       PROD
NATS       TEST     R                 R
******************************** Bottom of data ********************************
```

The above Flag settings would be specified for the following situation:

- Level Structure is TEST-CONS-HOLD-PROD
- The TEST level is an editing level and is not a complete Natural Environment. Modules are deleted from this level when promoted, and for that reason the **Keep Memb** flag is blank for the Test level.
- Promoting from TEST to CONS can be done either in a Set, or online using the “P” Operation. Either way, a job will be submitted to do the Load and Stow into the Natural Environment.
- Promoting to PROD can only be done in a Set, so Approval Rules (M.AR) should also be specified to force this to happen.

M.AD Natural Setup Entries

The Natural Interface uses the values specified in M.AD/NA to determine the Natural Environment to initiate, how to do this initiation, and what Natural Library to perform the operation against. An entry is Required per level.

Figure 7-2 shows the list of entries in M.AD/NA for an application called PLAY. Separate entries have been defined specifically for each level in this Application.

Figure 7-2. M.AD/NA Entries for PLAY Application

```
ISPW M.AD/NA       Natural Application Setup (PROD)                 Row 1 of 6
Command  ===>                                                  Scroll ===> CSR

List Commands: L Locate Entry, A Add an Entry
Line Commands: S Update Entry, D Delete Entry

  APPLLEVEL

  PLAYCONS
  PLAYHOLD
  PLAYPROD
  PLAYTEST
******************************** Bottom of data ********************************
```

Figure 7-3 shows the information required for a Natural application.

Figure 7-3. Natural Definition Information

```
ISPW M.AD/NA/S ---            NATURAL DEFINITION (PROD)
COMMAND ===>


 Applid (KEY) ==> PLAY
 Level  (KEY) ==> TEST      (TEST HOLD PROD etc.)


 Natural Library       ==>    PLAY-TST

 System Parm           ==>    PLYT
 Database Id           ==>    2


 Batch Invocation      ===>   NATBATT,NATPARM=',FSIZE=65'


 Online Invocation     ===>   SITENAT &SPARM
```

Site-specific Components and Install Notes

Sample Components

A small amount of customization is required for the Natural Interface to be installed properly. For reference purposes, sample and model components supplied in the ISPW installation SAMPLIB dataset may be reviewed to aid in the setup of Natural processing. These include the samples listed in Table 7-3.

Table 7-3. SAMPLIB Components for Setup of Natural Processing

Type	Name	Description
Panel	WZUGD	TEST level generate example
Panel	WZUGH	HOLD level generate example
Skeleton	WZUG	Generalized mainline “driver” skeleton
CLIST	WZUNAEXT	Unloads a Natural application from Natural into the Production PDS file(s)
JCL	WZUNALST	Creates a sequential file listing of all the Natural objects in a Natural Application
JCL	WZUNAUNL	Unloads the Natural objects listed in the output file from Job WZUNALST using CLIST WZUNAEXT
REXX	WZUNAAL	Allocate Natural datasets and start Natural online in TSO

Natural Dataset Allocation and Startup

The WZUNAAL REXX program is used to allow ISPW to invoke Natural. This REXX program allocates the Natural environment for the user when a Natural type module is accessed (where the action requires Natural) within an Assignment. If you already have a Natural startup REXX program, you can modify WZUNAAL to call it instead. Note, however, that WZUNAAL has a parameter passed which tells it which Natural load module to use. There is a TSO load module, which is used for interactive processing (the S for edit), and the batch load module, which is used for batched processing (the EP/IP for unload/load).

Copy the WZUNAAL CLIST to your site ISPW CLIST dataset. Modify it as described above to initiate a Natural environment for use within ISPW.

Unload of Natural Libraries into PDSs

ISPW manages the movement of the Natural components through PDSs. Thus, in Production, the source is stored in both the Production Natural library and the Production PDS. ISPW keeps these source copies in sync. Initially, when defining a new Natural application to ISPW, the Production Natural application library will have to be unloaded. Three SAMPLIB components are available to help do this. The first is JOB WZUNALST. This job creates a list of all the Natural components in a Natural library. To run it, change the jobcard, the Proc called, the Natural library name (in 2 places), and the output sequential dataset where the listing is to be stored (in 2 places). This dataset is the input to the unload step in JOB WZUNAEXT. To run this job, change the jobcard, the libraries to run the batch TSO session including Natural load libraries, DB2 libraries if required, and the ISPW CLIST library. Also change the parameters passed to the CLIST, providing the information required to start up the Production Natural region and unload from the correct library.

Chapter 8.
ISPW Web Interface

ISPW provides an optional web-based interface that supports a set of REST APIs that can be used to request ISPW functions. The interface also supports callbacks to the REST APIs supported by other products, in order to notify them when requested functions complete or fail.

Note: For a complete description of the ISPW Web Interface REST APIs, refer to the online Help for the Web Interface.

Required Components

The Web Interface is an optional feature that makes use of components of:

- CES (Compuware Enterprise Services)
- CMSC (Compuware Mainframe Services Controller).

To configure the CMSC, refer to the *Enterprise Common Components Installation and Customization Guide, MVS Version* for Release 17.02 and above.

To configure the CES, refer to the *Compuware Web Products Installation and Configuration Guide* for Release 17.02 and above.

Enabling Web Interface API Support

Once you have the required releases of CES and CMSC installed, you can enable ISPW's Web Interface REST API support by adding `WEBAPI=Y` to the SDEFINI parameter member used by the ISPW/CM task. For more information, see the chapter entitled "Install Base Software" in the *ISPW Installation and Configuration Guide*.

Note: Before setting WEBAPI=Y, ensure that the ISPW CM task JCL includes the DD statements for ETPLIB, ETPLOG, WQPLIB, and WQPLOG. See the ISPWCM member in SAMPLIB for an example of these DD statements.