AUTHORIZED DOCUMENTATION

XPOZ 6.1 Reference Guide

Novell®
Identity Manager Resource Kit

1.2

August 29, 2008

www.novell.com

Legal Notices

Novell, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc., has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the Novell Legal Patents Web page (http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

This is a reference guide to provide the information required to use the XPOZ test harness. XPOZ is the tool used to run tests for the Resource Kit.

Audience

This guide is intended for administrators who want to test their Identity Manager solution.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Documentation Updates

For the most recent version of the *XPOZ Reference Guide*, visit the Novell Compliance Management Platform Documentation Web site (http://www.novell.com/documentation/ncmp10/).

Additional Documentation

For documentation on Identity Manager, see the Identity Manager Documentation Web site (http://www.novell.com/documentation/idm36/index.html).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux* or UNIX*, should use forward slashes as required by your software.

What Is XPOZ?

1

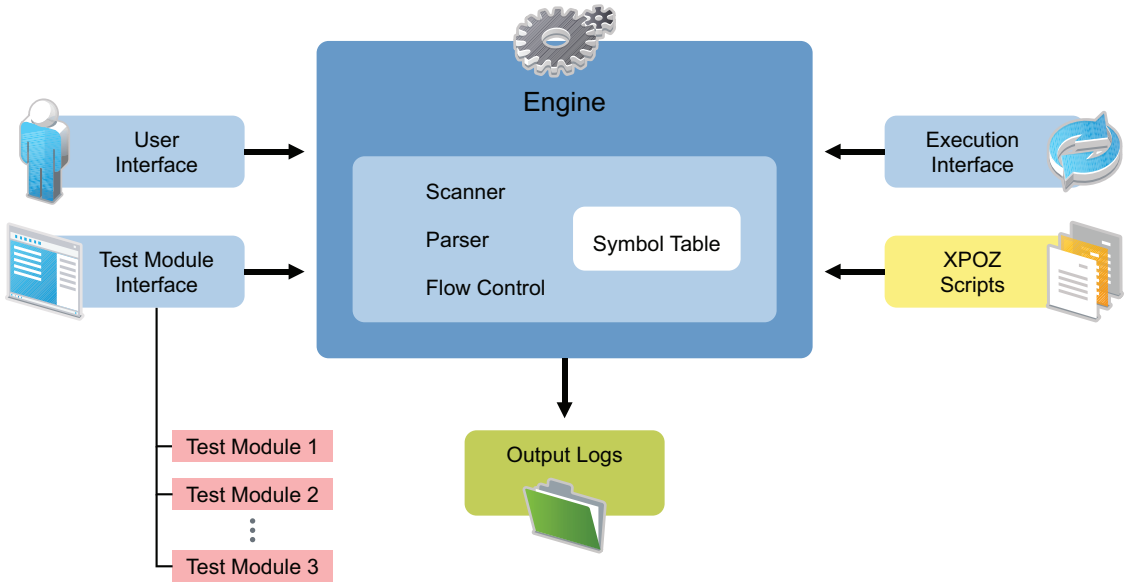
XPOZ, pronounced expose, is a script interpreter that provides access to a number of applications through their exported API interfaces. Originally it was developed as a test tool for Novell® Directory Services® and now has been expanded to provide a test harness for Identity Manager. XPOZ provides a solid foundation from which to build test modules. It can be extended by using native files (`.dll`, `.nlm`, `.so`) or Java* files (`.jar` or `.class`) to create test modules.

Using XPOZ for testing gives the following benefits:

- Faster turnaround time than compiled tests, because it is script driven.
- Scripts are easier to write than compiled tests.
- Better reuse of scripts.
- Easily supports new functionality without affecting existing tests and scripts.
- Minimal test tool memory footprint by only loading the test modules needed for a specific test.

XPOZ contains three major components: the engine, test modules, and scripts. The engine processes the XPOZ scripts and then passes that information into the test modules to perform the required task.

Figure 1-1 *XPOZ Architecture*

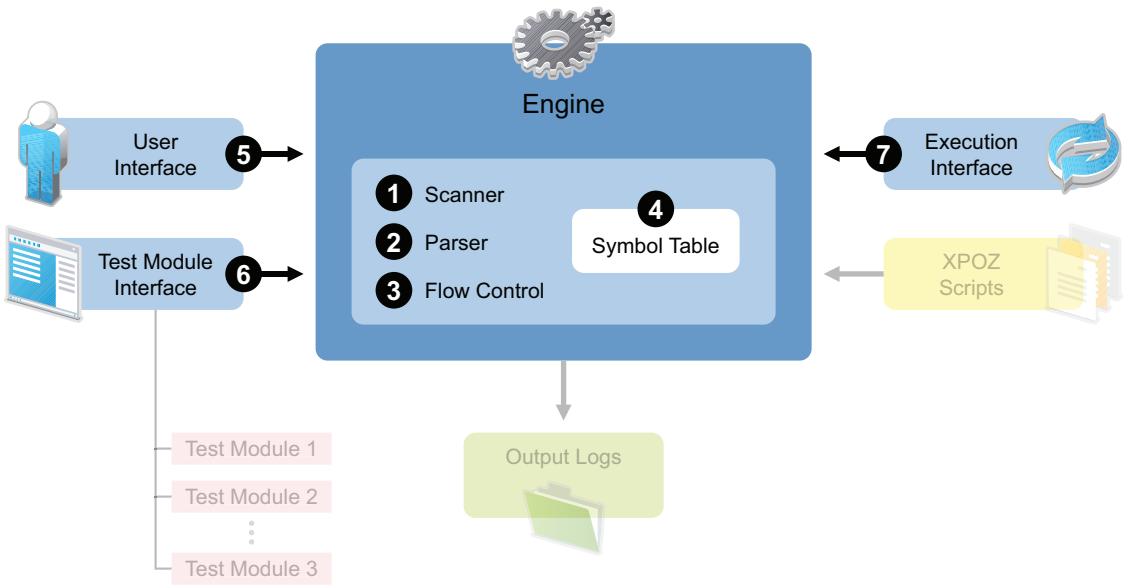


- Section 1.1, “Engine,” on page 10
- Section 1.2, “Test Modules,” on page 10
- Section 1.3, “XPOZ Scripts,” on page 11

1.1 Engine

The XPOZ scripts are accessed through the user interface or through the execution interface to the engine. The engine processes the XPOZ scripts to the next function or action, which it then passes into the test modules that perform the action.

Figure 1-2 *XPOZ Engine Components*



The engine is composed of seven separate items.

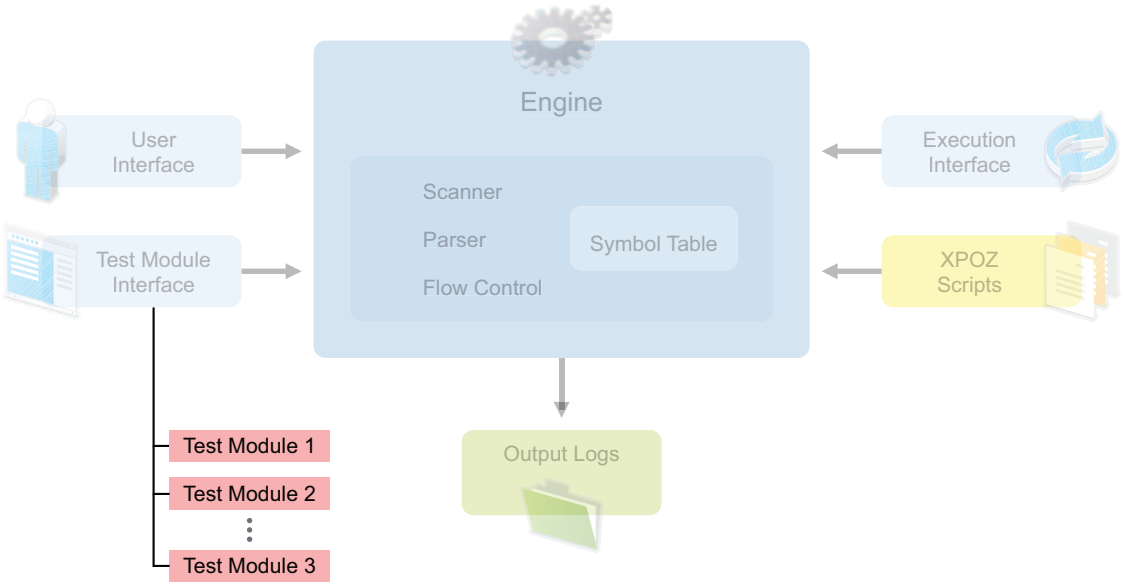
1. **Scanner:** Built upon the Purdue Compiler Construction Tool Set (PCCTS) architecture, this module tokenizes the scripts.
2. **Parser:** Built upon the PCCTS compiler architecture, this module interprets the tokens.
3. **Flow Control:** Determines the execution sequence of the test operations. Includes branching and looping constructs.
4. **Symbol Table:** Holds script-defined variables of multiple data types. Allows for multiple levels of variable scope.
5. **User Interface:** Script invocation point. Includes both command line and graphic (GUI) utilities.
6. **Test Module Interface:** Natively compiled extensions that provide script-callable functions that dynamically load test modules as needed by the script.
7. **Remote Execution Interface:** Allows remote execution of certain functions on remote systems. It requires the rcmdserver process to be running on the remote system.

1.2 Test Modules

Use test modules to execute the XPOZ scripts test. For example, if you need to test a move, create test module to represent a move of a user object in Lotus Notes*. Modules can simplify interactions with different systems.

The test modules are written in Java or C languages.

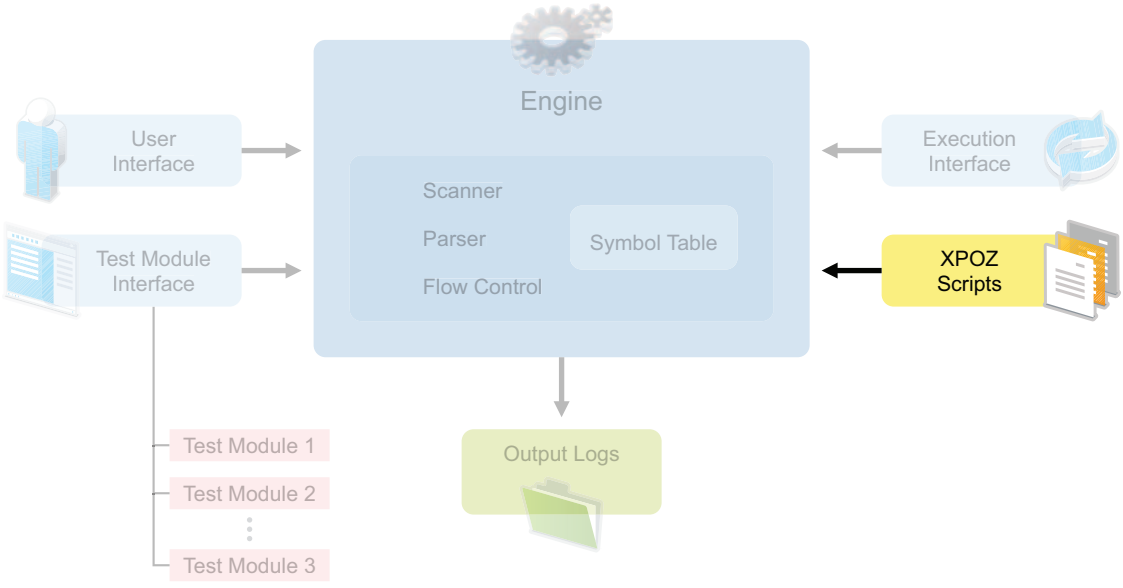
Figure 1-3 *XPOZ Test Modules*



1.3 XPOZ Scripts

You create XPOZ scripts to perform actions within different systems, allowing the script writer to generate administrative tasks in a testing environment. In an Identity Manager environment, scripts allows you to verify an add in Active Directory to an Identity Vault, or to any other connected system. For more information about the XPOZ scripts, see Chapter 4, “XPOZ Scripting Language,” on page 17.

Figure 1-4 *XPOZ Scripts*



Installing XPOZ

2

You can install XPOZ on either Linux or Windows*.

2.1 Prerequisites

The only prerequisite is to have NICI (Novell International Cryptographic Infrastructure) installed. You must verify that NICI is installed on the machine before installing XPOZ. If you don’t have NICI installed, follow the instructions in the NICI Administration Guide (http://www.novell.com/documentation/nici27x/pdfdoc/nici_admin_guide/nici_admin_guide.pdf) to install NICI.

To verify that NICI is installed on Windows:

1. Access the Control Panel.
2. Select *Add or Remove Programs*.
3. Verify there is an entry for NICI, then close the Add or Remote Programs window and the Control Panel.

To verify NICI is installed on Linux/UNIX:

1. Log in as `root`.
2. Enter `rpm -qa | grep -i nici`

2.2 Installing XPOZ on Linux/UNIX

1. Download the `xpozv61_install.zip` file that contains the installation files for XPOZ from the Novell Resource Kit 1.2 Download Web site (http://download.novell.com/index.jsp).
2. Extract the file to access the installation files.
3. Log in as `root`.
4. Change to the directory that contains the installation file, then enter:

   ```
   ./xpoz_install_linux.bin
   ```
5. Press Enter until the end of the license agreement, then enter `Y` to accept the license agreement.
6. Enter the installation location.

   The default location is `/usr/novell/xpoz`.
7. Review the installation location, then press Enter to start the installation.

2.3 Installing XPOZ on Windows

1. Download the `xpozv61_install.zip` file that contains the installation files for XPOZ from the Novell Resource Kit 1.2 Download Web site (http://download.novell.com/index.jsp).

2 Change into the directory that contains the installation files, then double-click `xpoz_install_windows.exe` to start the installation.

3 Use the following information to complete the installation:

- **Install Location:** Specify the installation location for XPOZ. The default location is `c:\Program Files\Novell\XPOZ`.

Upgrading XPOZ

3

3.1 What’s New

3.1.1 Results System

XPOZ 6.1 has a results system to display the test results in a Web page. For more information, see Chapter 6, “Configuring the Results to Display,” on page 31.

3.1.2 Added New Modules

XPOZ 6.1 has the following new modules to increase the functionality of XPOZ:

- NICI
- SAP
- GroupWise
- Telnet
- NT (Domain and Registry calls)
- Exchange 5.5 (DAPI)
- PKI
- JVM Module (provides the ability to call into java modules)
- Notes
- ODBC (Windows only)

There are also additional functions to existing modules, including:

- GetPlatformInformation
- XMLWriteParsedData
- NCPVRCacheUtil
- NCPVRQueueEvent
- NCPVRGetDriverStats
- NCPVRResetDriverStats
- NCPVRGetDirXMLVersion
- NCPVRGetPasswordsState

- NCPVRGetReciprocalAttrMap
- LDAPNMASGetPasswordStatus

3.1.3 XPOZ Function Documentation

The changes to the XPOZ function documentation are:

- Updated and modified all of the XPOZ function documentation for consistency.
- Modified the documentation to provide conditional tags.

The XPOZ function documentation is available at the Resource Kit Product Download Web site (http://download.novell.com/index.jsp).

3.2 Upgrading XPOZ

To upgrade XPOZ, follow the installation procedure. There is no separate upgrade procedure. For the installation procedure, see Chapter 2, “Installing XPOZ,” on page 13.

XPOZ Scripting Language

4

The XPOZ scripting language is comparable to the C language, although not exactly equivalent to it. XPOZ scripts contain the calls to XPOZ test functions. There are three elements found in the XPOZ scripts: variable declarations and assignments, flow control constructs, and test function calls.

Script files can call other script files. Upon completion of a script file, execution passes back to the calling script file, if one exists. Otherwise, the test terminates.

Variable scope is defined to be at the script and block level. All variables declared inside a curly brace block are thrown away upon the exit of the block. Any variables defined at the script level are thrown away when the script file exits. Specifically, this means that a variable x defined in script1 is visible during execution of script1 along with all other script files that script1 calls. Variable x goes out of scope (effectively is removed from the symbol table) when script1 completes. It is permissible to redefine variable x at a subordinate level (that is, in script2, which is invoked by script1). In this case, during the scope of script2, the local variable x is used instead of the variable defined in script1. Within a given script, a variable name can be defined only once.

4.1 Basic Variable Data Types

XPOZ incorporates a set of foundational data types for variables. They are:

- int
- unsigned int
- long
- unsigned long
- string
- char
- boolean

A variable can have any alphanumeric name, if it begins with an alphabetic character or an underscore. All XPOZ commands, including variable declarations and assignments, are separated by semicolons.

Some examples of variables are:

```
int i=0;
string thisIsAString = "A long string value goes here";
uint x=490000;
char middleInitial = 'B';
boolean someFlag=false;
```

4.2 Complex Variable Data Type

XPOZ allows single dimension arrays for any of the fundamental data types. These arrays can be accessed by using an index into the array, as is done in the C language. Square brackets [] denote an array. Arrays are 0-relative.

Some examples are:

```
int ageArray[] = { 35, 24, 80, 11, 7 };

if(ageArray[4] == 7) { ....}
```

4.3 Private Variables

The use of `private` before any variable declaration hides the value within the GUI. However, private variables can be accessed and displayed through the test modules, so they don't guarantee security for data.

4.4 System Variables

XPOZ defines a set of system variables. These have special meaning and can be used to enhance a test's interpretation and usability.

Table 4-1 *XPOZ System Variables*

Variable	Type	Description
Title1	string	Text to be printed in the GUI upper dialog first field. Default: NULL.
Title2	string	Text to be printed in the GUI lower dialog first field. Default: NULL.
ScriptFile	string	Filename of the currently executing script.
ScriptDir	string	File system directory from which to initiate a script. Default: "."
LogFile	string	Name of the current log file.
LogDir	string	File system directory in which to place LogFile.
RetryDelay	int	Number of seconds to wait before retrying a failed function call. Ignored if RetryMax is <=0. Default: 0

Variable	Type	Description
RetryMax	int	Number of times to retry a failed function call before failing the test. Default: 0
RetryMaxFlag	int	0: (default) Does not prompt the user on failed function calls, 1: Prompt user on failed function calls.
LogToFileFlag	boolean	False: Do not log to the output file. True: Log to output file. Default: True.
LogToScreenFlag	boolean	False: Do not log to the screen. True: Log to screen. Default: True.
TestCase	string	Marker to delineate the beginning of a new test case. The value is used as the name of the test case. Default: NULL.
TestCaseEnd	string	Marker to delineate the end of a test case. Default: NULL. The start of a new TestCase automatically implies the end of the previous TestCase.
TestDefID	string	Name of the test definition associated with this script.
TestName	string	Name of the test. Placed in the upper status bar. Default: NULL. When the results are turned on, this tag starts the script execution.
NotYetImplemented	int	Specifies that the next test case defined is not fully implemented. This prevents an incomplete test from counting as either a pass or fail within a script.
LogOnRetryFlag	boolean	False: Do not log retries to the display True: Log retries to the display. In the GUI, the retry counts are recorded in the lower status area. This flag attempts to clean up the display output area of the GUI. Default: False.
ReleaseName	string	Data is printed in upper dialog second field. Default: NULL.
rcmdSingleThreaded	int	Causes all calls made to RCMD to run on same thread. Default: 0

4.5 String Concatenation

XPOZ allows a shortcut way of appending strings. The + character is interpreted to instruct that two strings be concatenated together. The strings can be literals or other string variable names.

Some examples are:

```
string str1 = "This is a test " + "of concatenating strings " + "together";
string str2 = str1 + " - second appendage";
```

4.6 Special Script Codes: Embedding Hex Codes and Unicode

XPOZ allows integer values to be entered as hex codes. This is done by prepending 0x to the hex code. For example, 0x20 is equivalent to the integer 32. The script assignment looks like this:

```
int privileges = 0x20;
```

XPOZ also provides for using Unicode* strings in scripts. This is done by having the first two characters of the string be 0x. Everything following the 0x must be valid Unicode characters, with the low-order byte first. A script assignment looks like this:

```
inputStr = "0x43004e003d00fe34f0fefff9a7fffe792e00";
```

4.7 Special Script Codes: Comments

XPOZ provides two methods of denoting comments: single-line and multi-line. The single-line comment is noted by a double forward slash (//). The rest of the line is considered a comment and is ignored by the interpreter.

A multi-line comment begins with a forward slash-asterisk (/*) and ends with an asterisk-forward slash (*/). Everything between the two markers is considered part of the comment and is ignored by the interpreter.

Single-line comments can be embedded within multi-line comments.

4.8 Log Files

All script output is sent to the screen, if enabled, and is potentially sent to a log file. Proper usage of the LogFile, LogDir, and LogToFile variables determines if and when output is sent to a log file. For the Results Tracking System, if the file specified to be used already exists, the filename is modified based on the Test Definition.

The file format for XPOZ log files is HTML. This allows for easy browsing via a Web browser. All coloration notations are maintained with this approach.

The log file name (variable LogFile) can be changed from within a script. This is occasionally beneficial when it makes sense to isolate and report on a subset of the script separately from the rest of the script.

4.9 Mandatory Variables

XPOZ provides a special function that checks to assure that certain variables are defined prior to test execution. If all specified variables are not found in the symbol table, execution pauses with a message informing the user which variables must be included for test execution to progress. This stops a test from proceeding for possibly hours into a script before finding out that a required variable is missing.

The format for this function has one repeatable parameter: variable. The value for each variable parameter is a name-description pair. In the following example, SRV1 is a required variable, and its description is fully qualified First Server Name.

```
CheckMandatoryVariables(
  Variable = {"BootStrapAddress", "Address of a server with port in the tree."},
  Variable = {"SRV1", "Fully-qualified First Server Name"},
  Variable = {"SRV2", "Fully-qualified Second Server Name"},
  Variable = {"SRV1_Platform", "Platform of first server"},
  Variable = {"SRV2_Platform", "Platform of second server"},
  Variable = {"Admin", "Fully-qualified Admin Object Name"},
  Variable = {"AdminPassword", "Password for admin object"});
      Private
```

4.10 Environment Scripts

As noted previously, XPOZ scripts can invoke other XPOZ scripts. The difference between an `.xpoz` and a `.env` script is that the `.env` script does not generate a new execution level. Otherwise, these environment scripts behave like regular XPOZ scripts and are used to define the environment needed to run the rest of the script.

Often, the `.env` file can be thought of as a test wrapper. It sets up a test environment, places values on variables used by multiple test scripts, and then invokes a series of test scripts. Using `.env` files in this manner allows for easy combination of test script files without setting up the test environment for each test script.

4.11 XPOZ Test Functions

The real test work is accomplished in test functions. The scripts call these functions with certain parameters and expected values. The functions contain the code that links to external libraries to accomplish an objective. Within the test function is the check for whether the expected result was achieved or not. This allows the script to check for both positive and negative results. Often, an XPOZ test function is a wrapper for a third-party library function, adding to it the ability to check results along with the chance of retrying the function call for a loosely consistent database.

A loosely consistent database is one where synchronization is involved, so it is possible that a check is made prior to completing synchronization. In these cases, the check should be repeated for a predetermined number of times before failing the test.

The script format for an XPOZ test function is:

```
<test function name>(set of parameter name-parameter value set entities);
```

You can also include the optional `<return code>` at the front of the test function. The return code is always an integer and can be checked by the XPOZ script and subsequently used to determine script flow control.

All XPOZ test functions use a common parameter called `ExpResult`. The `ExpResult` parameter is a multivalued entity that describes what the result from the test is expected to be. Often a test function can return any one of a set of values and still be considered successful. For example, if a script calls a function to add an object, a successful answer would either be “object added” or “object already exists”. The `ExpResult` parameter allows you to specify all acceptable results. If any of them is found, the call is considered a success.

Additional values for the `ExpResult` parameter include:

- “OK”
- “BAD”
- “GREATER_THAN_ZERO”
- “Y”
- “YES”
- “ZERO_OR_GREATER”
- “TRUE_OR_FALSE”
- “PASS_THROUGH”

The documentation for the XPOZ functions is available for download on the Resource Kit download page.

4.12 Case Sensitivity in XPOZ Scripting

Script variables in XPOZ are case sensitive. Parameter names in test functions are not case sensitive, with one exception. If a parameter name in a different uppercase/lowercase configuration is equivalent to an XPOZ variable, the parameter name case is significant.

4.13 XPOZ Script Flow Constructs

Declarations instantiate variables for use within a script. They consist of a data type, a variable name, and an optional assignment operator and variable value. Examples include:

```
int iterator = 0;
int iteratorA = 0;
char middleInitial = 'b';
string companyName = "Acme, Inc.";
```

White space between operators and other tokens (for example, variable names, and values) is optional, but suggested for script readability.

A statement is either a single expression, or a block of statements. A single expression generally follows the C-language constructs, and allows the following operators:

Table 4-2 *XPOZ Single Expression*

C-Language Construct	Operators
assignment	=, +=, -=, *=, /=
logical AND	&&
exclusive OR	^
equality	==, !=
additive	+, -
unary (pre/post)	++,--
not	!
logical OR	||
bitwise OR	|
bitwise AND	&
relational	<, <=, >, >=
multiplicative	*, /
unary (post)	++, --

A block of statements encapsulates a set of statements with curly braces ({, }). Most often a block of statements is used in branch and loop constructs.

4.13.1 Branch and Loop Constructs

Branch Constructs

XPOZ allows two branch constructs: if-then-else and switch-case. The switch-case construct uses the break command to exit once it executes a desired branch.

If-Then-Else Statement Format

The else portion is optional:

```
if (boolean statement) {
  statement block
}
[else {
  statement block
}]
```

Switch-Case Statement Format

```
switch(integer variable) {
  case <integer value>: statement block; [break;]
  ...
  case <integer value>: statement block; [break;]
  default: statement block [break;]
}
```

Loop Constructs

XPOZ allows the following loop constructs: for, do-while, and while. The `continue` and `break` commands are allowed within loop constructs. The `continue` command ends the current iteration of the loop and proceeds with the following iteration. The `break` command ends the loop construct completely and continues with the next statement.

For Statement Format

```
for([variable initialization]; boolean statement; [variable modification,]) {
  statement block
}
```

Do-While Statement Format

```
do {
  statement block
} while(boolean statement);
```

While Statement Format

```
while( boolean statement) {
  statement block
}
```

A complete description of the XPOZ grammar (in BNF format) is included in Appendix A, “XPOZ Grammar,” on page 35.

Return and Exit Commands

The `return` and `exit` commands can be used anywhere in a script. The `return` command ends the current script and returns to the calling script. The `exit` command ends the entire test.

Sample Script

The following is a short XPOZ script sample showing how an XPOZ script is called from another XPOZ script. Also, variable scope is shown.

```
FILE_1.xpoz
// *********************
// Declaring local variables
// *********************
string str1 = “This is FILE_1 string one”;
int inc;
for(inc=0; inc<5; inc++) {
  PrintToScreen(String = inc + “\n”);
}
PrintToScreen(String = str1 + “\n”);
PrintToScreen(String = str2 + “\n”);
//END OF FILE_1


FILE_2.xpoz
// *********************
// Declaring local variables
// *********************
string str1 = “This is FILE_2 string one - longer”;
string str2 = “This is FILE_2 string two”;
string str3 = “This is FILE_2 string three”;
string str4 = “This is FILE_2 string four”;
string str5 = “This is FILE_2 string five”;

length = Length( String = str1);
PrintToScreen( String = "[" + length + "]  " + str1 + "\n");

//Calling FILE_1.xpoz from FILE_2.xpoz
SubFile(FILE = "FILE_1.xpoz");
//END OF FILE_2
```

The output of executing `FILE_2.xpoz` is:

```
[34]  This is FILE_2 string one - longer
0
1
2
3
4
This is FILE_1 string one
This is FILE_2 string two
```

Executing XPOZ Tests

5

After XPOZ is installed and you have created XPOZ scripts, you can run them to test the functionality of the drivers. There are multiple interfaces that allow you to run the scripts once they are implemented.

5.1 XPOZ Console

The XPOZ Console is a command line utility. Architecturally it is identical to the XPOZ GUI; however, the GUI is more flexible and powerful than the XPOZ Console. On Linux, the console view can be executed remotely via Telnet or ssh.

To run the XPOZ Console, enter:

```
XPOZConsole scriptfile [localEnvironmentFileName]
```

Linux is case sensitive and Windows is not. The `scriptfile` is the name of the XPOZ script that you want to execute. The *localEnvironmentFileName* allows you to pass in an environment file if it is needed; otherwise, it defaults to `XPOZ_ENV.txt`.

5.2 XPOZ GUI

The XPOZ GUI is a graphical utility that executes the XPOZ scripts. With it you can change variables during script execution and view their results, press buttons to manipulate execution, and have symbol table access.

To run the XPOZ GUI utility from the default location, use the following command:

- **Windows:** `c:\Program Files\Novell\XPOZGui.exe`
- **Linux:** `/usr/novell/XPOZGui`

Figure 5-1 *XPOZ GUI*

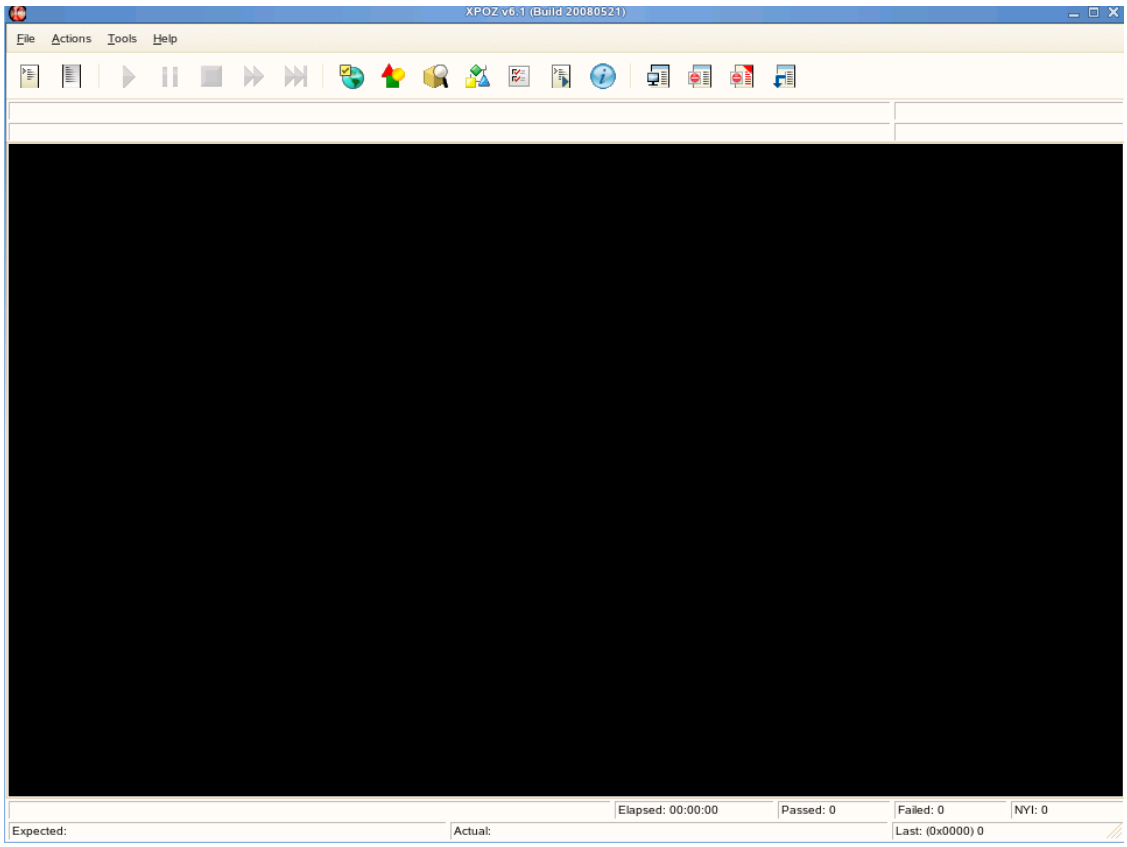


Table 5-1 describes the different features of the XPOZ GUI tool. The browsers allow you to view specific information to see why a script failed.

Table 5-1 *XPOZ GUI Features*

Feature	Description
Open Script File	Browse to and select the XPOZ script file to execute.
Open Log File	The feature is not implemented.
Play	Executes the XPOZ script.
Pause	Pauses or resumes the XPOZ script file execution.
Stop	Stops the execution of the XPOZ script file.
FFWD Retry	If errors occur during the execution of the XPOZ script, it skips the current delay before recalling the function.
FFWD Retry Count	Makes one more retry of the script before continuing with the execution.
Environment Browser	Browse to and modify an environment variable.
Symbol Table Browser	Browse to and modify a symbol defined in the symbol table.

Feature	Description
 Object Browser	Browse to a specific eDirectory™ object. The *Object Browser* lists each eDirectory tree you are authenticated to.
 Schema Browser	Browse to the eDirectory schema. The *Schema Browser* lists each eDirectory tree you are authenticated to.
 Options Browser	Modify common environment variables, such as log to screen or retrydelay.
 Execute Script Command	Inserts additional script commands into the XPOZ script that is running.
 Script Information Dialog	Lists the information about the script being executed.
 Results Admin	Allows you to access and reuse the results of the XPOZ script.
 Turn Results Off	Turns off the results. The results are no longer recorded.
 Turn Results On	Turns on the results. The results are recorded.
 Turn Defect Tracking Off	The feature is not implemented.
 Turn Defect Tracking On	The feature is not implemented.
 Select Results System	Allows you to select the type of results tracking system. Currently the only supported system is Results Tracking System v1 (RTS).

5.3 RCMD

RCMD is the execution interface that allows remote execution of certain functions on remote systems. The rcmdserver process must be running on the remote system for this to work. Use the following convention to enable remote calls:

```
[\\remoteaddress\]<funcname>(parms);
```

If the `remoteaddress` is the string `localhost`, then execution is only made locally.

Configuring the Results to Display

6

With XPOZ, you can store results from a test script into an LDAP object. XPOZ also contains a web interface that categorizes and displays the results. This allows for easy access to the results, and you can use the results for progress reports and trending.

6.1 Configuring XPOZ to Display the Results

1. Verify that you have a Web server with PHP 4 module installed and running.
2. Make sure you have access to an eDirectory™ tree.

   This can be the same server that is running the Metadirectory engine, or a new tree for results purposes. The results system uses LDAP, so any LDAP server should work with the results. However, the results have only been tested against the LDAP server on eDirectory. If you do use a different LDAP server, the XPOZ scripts must be converted to work against this LDAP server.
3. After XPOZ is installed, run the `RTSObjects-Schema.xpoz` XPOZ script to extend the schema before you run any tests.
4. Unzip the `rts.zip` file into the `htdocs` (or equivalent) directory on the Web server.
5. Add the following information for the results object in the `rts.cfg` file:
   - **IP address:** The IP address of the LDAP server.
   - **Port:** The cleartext LDAP port.
   - **Password:** The password for the user as specified by User ID.
   - **Base Container:** The container used in the XPOZ script in Step 3.
   - **User ID:** The DN of a user with read/write rights to the base container that is used in the XPOZ script in Step 3.

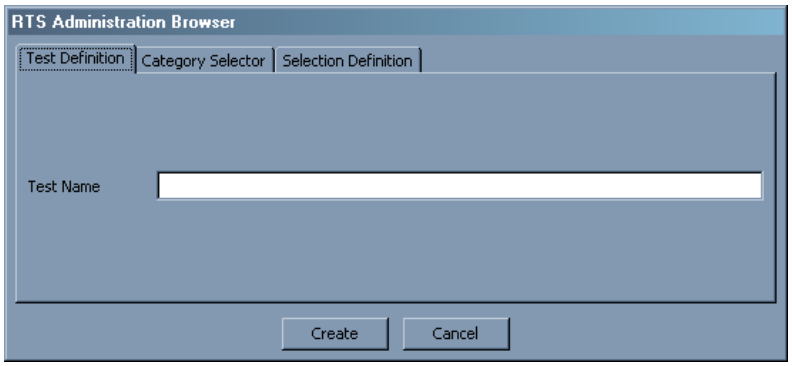
6.2 Creating and Managing the Results Objects

The results are stored as objects in an LDAP directory. These objects are used to sort, categorize, or define the test objects that are created. These objects, in conjunction with the test run objects, generate the results pages. The different objects that must be created are:

- **Test Definition:** A base test that corresponds to the `TestDetID` tag within a script.
- **Category Selector:** A title bar in the Web page used to separate test definitions within a selection definition so that they are organized. For example, your category selectors could be platforms (AIX, Solaris, Windows, or Linux) or components (Notes driver or Active Directory driver).
- **Selection Definition:** A list of all of the tests that are to be run against a given build.

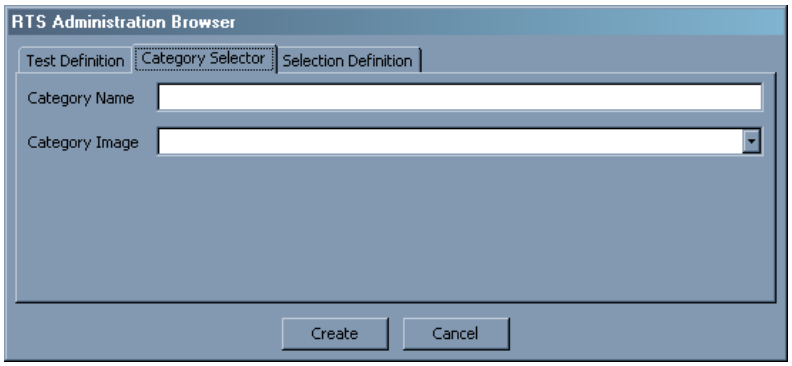
To create the results objects:

1. In the XPOZ GUI, click the *Results Admin* icon  .
2. Specify a name for the test, then click *Create*.



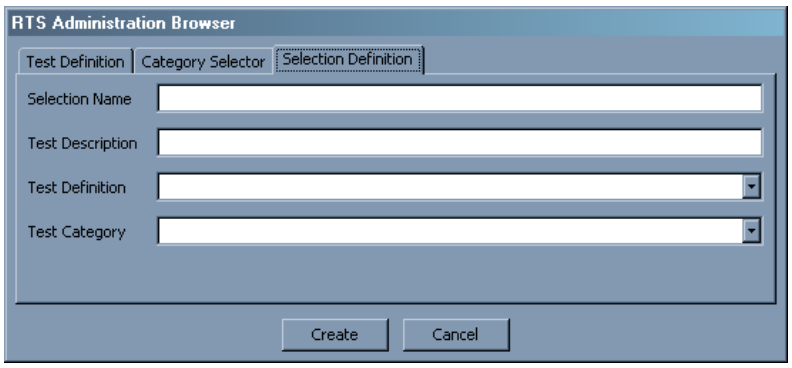
This creates a test definition and must be done for each unique test.

3. Click the *Category Selector* tab.
4. Specify a name for the category, then click *Create*.



This creates a category selector and must be done for each unique separator.

5. Click the *Selection Definition* tab.



6. Specify a selection name. This is the definition that this test run is placed under.
7. Specify a test description. This is the description that is seen from the Web view.

8 Select a test definition from the drop-down list. This is the test that runs when this definition runs.

9 Select a test category image from the drop-down list, then click *Create*.

This creates or modifies a selection definition and adds the specified test.

10 Repeat steps 7 through 9 to add additional tests to the same definition.

6.3 Enabling the Results in Each Script

For a test to start results tracking, you must specify the `TestName` tag at the beginning of the test. The value of the `TestDefID` tag corresponds to a test definition that is created on the results system. If you do not specify a `TestDefID` tag, you see all of the tests listed under this test definition.

You must also turn on results to send the information to the Web server.

1 In the XPOZ GUI, click the *Select Results System* icon  .

2 Select *Results Tracking System v1*, then click *OK*.

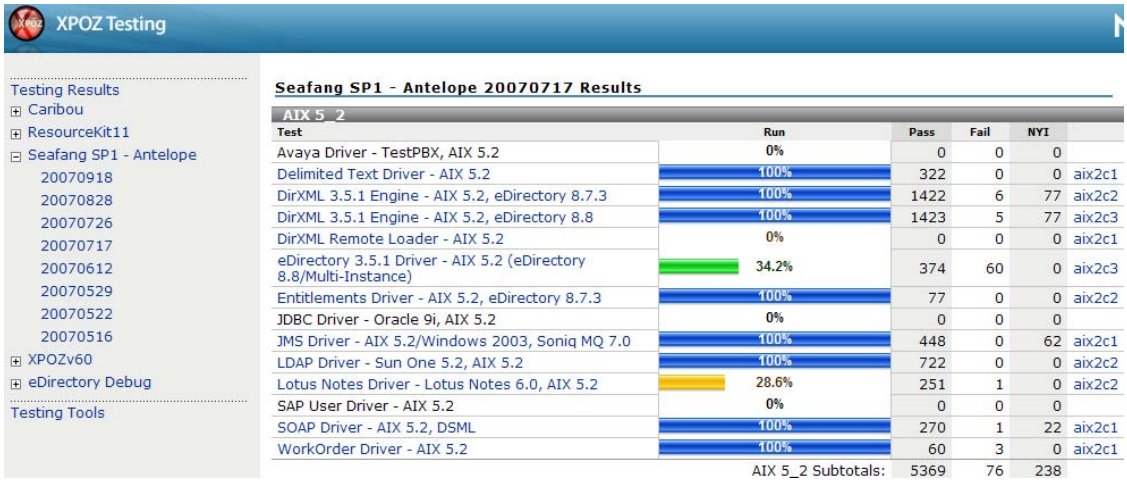
3 Click the *Turn Results On/Off* icon  to turn on results.

After the results are turned on and the script is executed, the information is placed in the LDAP directory. The Web server queries the directory to populate the Web page with the results information.

6.4 Web Page Layout

The Web page layout in Figure 6-1 is divided into multiple sections to make the results accessible and easy to read.

Figure 6-1 *XPOZ Results Web Page Layout*



- **Display Group:** A high-level category that contains builds for a given project. In the left frame of Figure 6-1, the display groups are Caribou, ResourceKit11, Seafang SP1 - Antelope, XPOZv60, and eDirectory Debug. In your environment, the display groups could be eDirectory 8.8 deployment or Identity Manager 3.5.1 regression. Selecting the display group gives an overview of all the builds.

- **Build:** Groups all of the tests that are run for a given project. There are multiple groups per display group. In Figure 6-1, the builds are the different dates for the Seafang SP1 - Antelope builds. The results for the selected display group are displayed in the table on the right.
- **Test Run:** Each line under a given build is a unique test run. The build level displays a high-level view of the statistics for each test as well as statistics within a category selector. In Figure 6-1 the category selector is AIX 5_2. The line is a link for more detailed information about each specific test that has been run.

XPOZ Grammar

A

The following is the actual XPOZ grammar in PCCTS format with actions and token definitions removed. Its form is BNF-like and demonstrates the recursive-descent nature of the parser. Items in all uppercase represent the token to be matched. Other identifiers in lowercase or mixed case correspond to a rule elsewhere in the grammar. Groups of tokens/rules enclosed in >{}= represent optional components. Groups enclosed in >( )= followed by a >+= indicate >one or more= of those groups and a >*= following a group represents zero or more of the groups.

```
declaration:
  {PRIVATE} DATATYPE IDENTIFIER { arraynotation } { ASSIGN initialize }
  ( COMMA IDENTIFIER { arraynotation } { ASSIGN initialize } )* SEMICOLON ;

block:
  LBRACE { combo_list } RBRACE;

combo:
  ( declaration | statement );

combo_list:
  ( combo )+ ;

statement:
  ( expression SEMICOLON
  | block
  | IF LPAREN expression RPAREN combo { ELSE combo }
  | WHILE LPARENexpression RPAREN combo
  | DO combo WHILE LPAREN expression RPAREN SEMICOLON
  | FOR LPAREN {expression } SEMICOLON { expression } SEMICOLON { expression }
RPAREN combo
  | SWITCH expression
    · LBRACE ( CASE expression COLON { combo_list } )* { DEFAULT COLON { combo_list
} } RBRACE
  | CONTINUE SEMICOLON
  | BREAK SEMICOLON
  | RETURN { expression } SEMICOLON
  | EXIT SEMICOLON
  | SEMICOLON
  );

expression:
  assignment_expression ( COMMA assignment_expression )* ;

assignment_expression:
  logical_OR_expression { OPERAND assignment_expression };

logical_OR_expression:
  logical_AND_expression ( LOGICAL_OR logical_AND_expression )* ;

logical_AND_expression:
  bitwise_OR_expression ( LOGICAL_AND bitwise_OR_expression )* ;

bitwise_OR_expression:
  exclusive_OR_expression ( BITWISE_OR exclusive_OR_expression )* ;
```

```
exclusive_OR_expression:
  bitwise_AND_expression ( BITWISE_XOR bitwise_AND_expression )* ;

bitwise_AND_expresion:
  equality_expression ( BITWISE_AND equality_expression )* ;

equality_expression:
  relational_expression ( EQUALITY relational_expression )* ;

relational_expression:
  additive_expression ( RELATIONAL additive_expression )* ;

additive_expression:
  multiplicative_expression ( ADDITIVE multiplicative_expression )* ;

multiplicative_expression:
  unary_expression ( MULTIPLICITIVE unary_expression )* ;

unary_expression:
  ( PLUS unary_expression
  | MINUS unary_expression
  | NOT unary_expression
  | INCREMENT unary_expression
  | DECREMENT unary_expression
  | primary_expression ( INCREMENT | DECREMENT | //Nothing )
  );

primary_expression:
  ( LITERAL_STRING_DELIM
  | LITERAL_CHARACTER_DELIM
  | number
  | identifier { LPAREN { argument_list } RPAREN }
  | LPAREN expression RPAREN
  | boolean
  );

argument_list:
  argument ((COMMA argument)* | SEMICOLON ( argument SEMICOLON )* ) ;

argument:
  IDENTIFIER ASSIGN initialize;

identifier:
  IDENTIFIER { arraynotation };

initialize:
  ( element_set | assignment_expression ) ;

element_set:
  LBRACE set_member ( COMMA set_member )* RBRACE;

set_member:
  ( element_set | logical_OR_expression );

number:
  ( OCT_NUM
  | HEX_NUM
  | INT_NUM
  | U_OCT_NUM
```

```
  | U_INT_NUM
  | U_HEX_NUM
  | L_OCT_NUM
  | L_HEX_NUM
  | L_INT_NUM
  ) ;

arraynotation:
  LBRACK ( assignment_expression ) RBRACK ;

boolean:
  ( TRUETOKEN | FALSETOKEN ) ;

start:
  ( declaration | statement )* ( XPOZEOF | . );
```

eDirectory Parameter Fields by Syntax

B

Some eDirectory™ syntaxes have multiple fields. XPOZ recognizes each syntax and expects certain fields for each syntax. The field order is significant. The fields required for each syntax are listed below:

```
SYN_BACKLINK:
  string remoteID
  string objName

SYN_BOOLEAN:
  int value (1=TRUE and 0=FALSE)

SYN_CE_STRING:
  string str

SYN_CI_LIST:
  int strTot (number of strings that follow)
  string String (occurs strTot times)

SYN_CI_STRING:
  string str

SYN_CLASS_NAME:
  string name

SYN_COUNTER:
  int value

SYN_EMAIL_ADDRESS:
  int type
  string address

SYN_DIST_NAME:
  string name

SYN_FAX_NUMBER:
  string phoneNumber
  int numberOfBit
  string bitString

SYN_HOLD:
  string objName
  int amount

SYN_INTEGER:
  int value

SYN_INTERVAL:
  int value
```

```
SYN_NET_ADDRESS:
  int addressType
  int addressLength
  string address (in hex)

SYN_NU_STRING:
  string str

SYN_OBJECT_ACL:
  string protectedAttrName
  string subjectName
  int privileges

SYN_OCTET_LIST:
  int strTot (number of string that follow)
  string str (occurs strTot times)

SYN_OCTECT_STRING:
  int length
  string str (‘alpha’ed hex, for example "AA")

SYN_PATH:
  int nameSpaceType
  string volName
  string path

SYN_POSTAL_ADDRESS:
  int strTot (number of addStr (6 max) that follow)
  string addStr

SYN_PR_STRING:
  string str

SYN_REPLICA_POINTER:
  string SrvrName
  int replType
  int replNumber
  int count
  NOTE: The following occur Count times, in "Type;Addr; Type;Addr" order. This
represents the "Hint" portion of the replica pointer.
  int networkType
  int netAddress

SYN_STREAM:
  int length (default 0)
  string data (default NULL)
  NOTE: While proper use of stream attributes does not require a value to be
specificed, it is allowed here so that improper use can be tested.

SYN_TEL_NUMBER:
  string String

SYN_TIME:
  int month
  int day
  int year 1900, etc
  int hour
  int minute
  int second
```

```
SYN_TIMESTAMP:
  uint wholeSecond
  uint eventID

SYN_TYPED_NAME:
  string objName
  int level
  int interval

SYN_UNKNOWN:
  string attrName
  int syntaxID
  string value
```

XPOZ Specific Error Codes

C

The following table shows the error codes defined by XPOZ. The text name of the error code should be used in the scripts instead of the error number.

Table C-1 *XPOZ Error Codes*

Error Number	Text Name
-2000	COMMON_ERR_BASE
0	ERR_NOERROR
COMMON_ERR_BASE -1	ERR_NOMEM
COMMON_ERR_BASE -2	ERR_DUPSYM
COMMON_ERR_BASE -3	ERR_INVALID_LEVEL
COMMON_ERR_BASE -4	ERR_NOT_FOUND
COMMON_ERR_BASE -5	ERR_SYS_ERROR
COMMON_ERR_BASE -6	ERR_BAD_TAG
COMMON_ERR_BASE -7	ERR_TAG_NOT_FOUND
COMMON_ERR_BASE -8	ERR_NO_VALUES
COMMON_ERR_BASE -9	ERR_EXPIRED_TIME
COMMON_ERR_BASE -10	ERR_SYNTAX_ERROR
COMMON_ERR_BASE -11	ERR_FAILED_TEST
COMMON_ERR_BASE -12	ERR_NOT_INITIALIZED
COMMON_ERR_BASE -13	ERR_SYMBOL_NOT_FOUND
COMMON_ERR_BASE -14	ERR_ILLEGAL_ASSIGN
COMMON_ERR_BASE -15	ERR_TYPE_MISMATCH
COMMON_ERR_BASE -16	ERROR_INVALID_TYPE
COMMON_ERR_BASE -17	ERR_BAD_UNICODE
COMMON_ERR_BASE -18	ERR_FREEMEM
COMMON_ERR_BASE -19	ERR_SIZE_MISMATCH
COMMON_ERR_BASE -20	ERR_USER_ABORT
COMMON_ERR_BASE -21	ERR_FAILED_MATCH
COMMON_ERR_BASE -22	ERR_MODLOAD_FAIL
COMMON_ERR_BASE -23	ERR_IMPFILE_INVALID
COMMON_ERR_BASE -24	ERR_ADDMOD_FAILURE

Error Number	Text Name
COMMON_ERR_BASE -25	ERR_MOD_NOT_FOUND
COMMON_ERR_BASE -26	ERR_IMP_SYMBOL
COMMON_ERR_BASE -27	ERR_UNIMP_SYMBOL
COMMON_ERR_BASE -28	ERR_IN_USE
COMMON_ERR_BASE -29	ERR_FUNCTION_INVALID
COMMON_ERR_BASE -30	ERR_MODUNLOAD_FAIL
COMMON_ERR_BASE -31	ERR_INVALID_TREE
COMMON_ERR_BASE -32	ERR_MAX_CX_REACHED
COMMON_ERR_BASE -33	ERR_XPOZCONN_TIMEOUT
COMMON_ERR_BASE -34	ERR_REC_NOT_FOUND
COMMON_ERR_BASE -35	ERR_FAILED_CONVERSION
COMMON_ERR_BASE -36	ERR_FUNCTION_NOT_FOUND
COMMON_ERR_BASE -37	ERR_FX_NOT_REGISTERED
COMMON_ERR_BASE -38	ERR_INVALID_SYMNAME
COMMON_ERR_BASE -39	ERR_INVALID_SUBSCRIPT
COMMON_ERR_BASE -40	ERR_INDEX_OUT_OF_RANGE
COMMON_ERR_BASE -41	ERR_FAILED_CREATE_SOCK
COMMON_ERR_BASE -42	ERR_UNKNOWN_HOST
COMMON_ERR_BASE -43	ERR_MISMATCHED_TILDE
COMMON_ERR_BASE -44	ERR_OBSOLETE_FUNCTION
COMMON_ERR_BASE -45	ERR_INVALID_RETURN_TYPE
COMMON_ERR_BASE -46	ERR_FAILED_CONNECT
COMMON_ERR_BASE -47	ERR_APPTEST_ERROR
COMMON_ERR_BASE -48	ERR_INVALID_CONN
COMMON_ERR_BASE -49	ERR_BAD_OCTET
COMMON_ERR_BASE -50	ERR_SOCKET_ERROR
COMMON_ERR_BASE -51	ERR_IO_ERROR
COMMON_ERR_BASE -52	ERR_TREEWALK_ERROR
COMMON_ERR_BASE -53	ERR_RESOLVE_ERROR
COMMON_ERR_BASE -54	ERR_INSUFFICIENT_BUF
COMMON_ERR_BASE -55	ERR_INVALID_FILE_MODE
COMMON_ERR_BASE -56	ERR_OPENING_A_FILE

Error Number	Text Name
COMMON_ERR_BASE -57	ERR_NOT_YET_IMPLEMENTED
COMMON_ERR_BASE -58	ERR_BUF_TOO_SMALL
COMMON_ERR_BASE -59	ERR_UNKNOWN_CMD
COMMON_ERR_BASE -60	ERR_ARG_NOT_FOUND
COMMON_ERR_BASE -61	ERR_INVALID_ADDRESS
COMMON_ERR_BASE -62	ERR_INVALID_IPX_ADDRESS
COMMON_ERR_BASE -63	ERR_INVALID_IP_ADDRESS
COMMON_ERR_BASE -64	ERR_SHUTDOWN_NOW
COMMON_ERR_BASE -65	ERR_INSUFFICIENT_ARGS
COMMON_ERR_BASE -66	ERR_CHECKSUM_MISMATCH
COMMON_ERR_BASE -67	ERR_UNKNOWN_VERB
COMMON_ERR_BASE -68	ERR_INVALID_SUBFUNCTION
COMMON_ERR_BASE -69	ERR_UNKNOWN_FUNCTION
COMMON_ERR_BASE -70	ERR_BAD_REQUEST
COMMON_ERR_BASE -71	ERR_INVALID_FILEHANDLE
COMMON_ERR_BASE -72	ERR_UNSUPPORTED_VER
COMMON_ERR_BASE -73	ERR_EOF
COMMON_ERR_BASE -74	ERR_CLASS_NOT_INITIALIZED
COMMON_ERR_BASE -75	ERR_TOO_MANY_ADDRS
COMMON_ERR_BASE -76	ERR_FILE_SIZE
COMMON_ERR_BASE -77	ERR_NAMESVC_NOT_INIT
COMMON_ERR_BASE -78	ERR_INVALID_DIRHANDLE
COMMON_ERR_BASE -79	ERR_INVALID_HOST_FORMAT
COMMON_ERR_BASE -80	ERR_BAD_HOST_NAME
COMMON_ERR_BASE -81	ERR_FILE_SKIPPED
COMMON_ERR_BASE -82	ERR_TOO_MANY_THREAD_IDS
COMMON_ERR_BASE -83	ERR_NO_PACKETS
COMMON_ERR_BASE -84	ERR_CMDLINE_ARG_INVALID
COMMON_ERR_BASE -85	ERR_TRY_AGAIN
COMMON_ERR_BASE -86	ERR_REMOTE_ONLY_CMD
COMMON_ERR_BASE -87	ERR_INVALID_STATE
COMMON_ERR_BASE -88	ERR_OUT_OF_BOUNDS

Error Number	Text Name
COMMON_ERR_BASE -90	ERR_NOT_SORTED
COMMON_ERR_BASE-91	ERR_LOCKED
-2100	ERRNO_BASE
ERRNO_BASE -1	ERR_NOENTRY
ERRNO_BASE -2	ERR_ARGLIST_TOOBIG
ERRNO_BASE -3	ERR_EXEC_FORMAT
ERRNO_BASE -4	ERR_BAD_FILEHANDLE
ERRNO_BASE -5	ERR_NO_MEMORY
ERRNO_BASE -6	ERR_ACCESSDENIED
ERRNO_BASE -7	ERR_FILE_EXISTS
ERRNO_BASE -8	ERR_CROSSDEVICE_LINK
ERRNO_BASE -9	ERR_INVALID_ARG
ERRNO_BASE -10	ERR_FILETABLE_OVERFLOW
ERRNO_BASE -11	ERR_TOOMANY_OPEN_FILES
ERRNO_BASE -12	ERR_NO_SPACE
ERRNO_BASE -13	ERR_ARG_TOOLARGE
ERRNO_BASE -14	ERR_RESULT_TOOLARGE
ERRNO_BASE -15	ERR_RESOURCE_DEADLOCK
ERRNO_BASE -16	ERR_RESOURCE_INUSE
ERRNO_BASE -17	ERR_SERVER_MEMORY_ERROR
ERRNO_BASE -18	ERR_NOSERVER
ERRNO_BASE -19	ERR_WRONG_OBJECT
ERRNO_BASE -20	ERR_TRANS_RESTARTED
ERRNO_BASE -21	ERR_RESOURCE_UNAVAILABLE
ERRNO_BASE -22	ERR_BAD_HANDLE
ERRNO_BASE -23	ERR_NO_SCREEN
ERRNO_BASE -24	ERR_RES_UNAVAILABLE
ERRNO_BASE -25	ERR_NO_SUCH_DEVICE
ERRNO_BASE -26	ERR_BAD_MESSAGE
ERRNO_BASE -27	ERR_BAD_ADDRESS
ERRNO_BASE -28	ERR_IO
ERRNO_BASE -29	ERR_NO_DATA

Error Number	Text Name
ERRNO_BASE -30	ERR_STREAM_UNAVAILABLE
ERRNO_BASE -31	ERR_FATAL_PROTOCOL_ERROR
ERRNO_BASE -32	ERR_BROKEN_PIPE
ERRNO_BASE -33	ERR_ILLEGAL_SEEK
ERRNO_BASE -34	ERR_IOCTL_TIMEOUT
ERRNO_BASE -35	ERR_EWOULDBLOCK
ERRNO_BASE -36	ERR_EINPROGESS
ERRNO_BASE -37	ERR_EALREADY
ERRNO_BASE -38	ERR_ENOTASOCK
ERRNO_BASE -39	ERR_EDESTADDREQ
ERRNO_BASE -40	ERR_EMSGSIZE
ERRNO_BASE -41	ERR_EPROTOTYPE
ERRNO_BASE -42	ERR_ENOPROTOOPT
ERRNO_BASE -43	ERR_EPROTONOSUPPORT
ERRNO_BASE -44	ERR_ESOCKTNOSUPPORT
ERRNO_BASE -45	ERR_EOPNOTSUPP
ERRNO_BASE -46	ERR_EPFNSUPPORT
ERRNO_BASE -47	ERR_EAFNOSUPPORT
ERRNO_BASE -48	ERR_EADDRINUSE
ERRNO_BASE -49	ERR_EADDRNOTAVAIL
ERRNO_BASE -50	ERR_ENETDOWN
ERRNO_BASE -51	ERR_ENETUNREACH
ERRNO_BASE -52	ERR_ENETRESET
ERRNO_BASE -53	ERR_ECONNABORTED
ERRNO_BASE -54	ERR_ECONNRESET
ERRNO_BASE -55	ERR_ENOBUFS
ERRNO_BASE -56	ERR_EISCONN
ERRNO_BASE -57	ERR_ENOTCONN
ERRNO_BASE -58	ERR_ESHUTDOWN
ERRNO_BASE -59	ERR_ETOOMANYREFS
ERRNO_BASE -60	ERR_ETIMEDOUT
ERRNO_BASE -61	ERR_ECONNREFUSED

Error Number	Text Name
ERRNO_BASE -62	ERR_EBUSY
ERRNO_BASE -63	ERR_EINTR
ERRNO_BASE -64	ERR_EISDIR
ERRNO_BASE -65	ERR_ENAMETOOLONG
ERRNO_BASE -66	ERR_ENOSYS
ERRNO_BASE -67	ERR_ENOTDIR
ERRNO_BASE -68	ERR_ENOTEMPTY
ERRNO_BASE -69	ERR_EPERM
ERRNO_BASE -70	ERR_ECHILD
ERRNO_BASE -71	ERR_EFBIG
ERRNO_BASE -72	ERR_EMLINK
ERRNO_BASE -73	ERR_ENODEV
ERRNO_BASE -74	ERR_ENOLCK
ERRNO_BASE -75	ERR_ENOTTY
ERRNO_BASE -76	ERR_EFTYPE
ERRNO_BASE -77	ERR_EROFS
ERRNO_BASE -78	ERR_ESRCH
ERRNO_BASE -79	ERR_ECANCELED
ERRNO_BASE -80	ERR_ENOTSUP
ERRNO_BASE -81	ERR_ECANCELLED
ERRNO_BASE -82	ERR_ENLMDATA
ERRNO_BASE -83	ERR_EILSEQ
ERRNO_BASE -84	ERR_EINCONSIS
ERRNO_BASE -85	ERR_EDOSTEXTEOL
ERRNO_BASE -86	ERR_ENONEXTANT
ERRNO_BASE -87	ERR_ENOCONTEXT
ERRNO_BASE -88	ERR_ODBCCONNECT
ERRNO_BASE -89	ERR_SOCKET
-2200	ERR_EXTBASE
ERR_EXTBASE -1	ERR_FTPFAILED
ERR_EXTBASE -2	ERR_CWD_FAILED
ERR_EXTBASE -3	ERR_FAILED_DATACONN

Error Number	Text Name
ERR_EXTBASE -4	ERR_INVALID_SIZE
ERR_EXTBASE -5	ERR_FAILED_REMOTE_REQUEST
ERR_EXTBASE -6	ERR_FAILED_DELETE
ERR_EXTBASE -7	ERR_DIR_ALREADY_EXISTS
ERR_EXTBASE -8	ERR_NOREPLY
ERR_EXTBASE -9	ERR_NOFILES
ERR_EXTBASE -10	ERR_FAILED_AUTH
ERR_EXTBASE -11	ERR_INVALID_PASS
ERR_EXTBASE -12	ERR_COPY_FAILED
ERR_EXTBASE -13	ERR_CONN_FAILED
ERR_EXTBASE -14	ERR_MKD_FAILED
ERR_EXTBASE -15	ERR_RMD_FAILED
ERR_EXTBASE -16	ERR_RENAME_FAILED
ERR_EXTBASE -17	ERR_DELETE_FAILED
ERR_EXTBASE -18	ERR_SEND_FAILED
ERR_EXTBASE -19	ERR_NO_RESPONSE