Inter-System Call Exchange (ICE)

Subroutines Guide

Revision 7.2

Table of Contents

1 Overview

The Inter-System Call Exchange, ICE is a set of procedures built on the Network Process Exchange, NPX. NPX provides a general capability to make a remote procedure call to any dynamically linkable subroutine on a remote system.

On the user's first NPX call to a system, extensive security checks are performed to ensure only valid calls are accepted. Subsequent calls do not perform this validation check. The call is then passed to an NPX Slave Process. A slave is a PRIMOS process with the sole function of executing procedure calls at the request of remote users. When idle, a slave assigns a Primenet port, releases all resources such as wired memory and "hibernates," waiting for a call to come in. Each slave acts for a single remote "master" and remains assigned to that master until released. In this way, a master has an exclusive server on each of possibly several systems for the duration of its remote activity, thus providing a mechanism for implementing a "distributed" system.

The slave unpacks the subroutine name and parameters, builds a standard calling sequence and calls the procedure. The procedure, unaware that it is being executed on behalf of a remote user, performs its expected function. The slave will now transmit the results back to its master.

Throughout this operation, the user is unaware that any remote activity has occurred.

2 Purpose

NPX is an undocumented Prime tool, it is subject to change at any time. Using ICE will protect the NPX user from making extensive changes to their application. The first benefit will be seen at PRIMOS revision 19.3, the NPX calling interface has changed.

3 Implementation

As mentioned above, ICE is a layer above NPX. This method of implementation somewhat limits the functionality of ICE. As new NPX procedures become available they will be incorporated into ICE. However, because NPX is a layer above PRIMENET, it may be possible to implement some additional functionality not provided by NPX.

4 Performance

The NPX mechanism is extremely useful but has some performance drawbacks. The main concern is the amount of CPU required to pack the subroutine arguments, transfer the arguments, build the procedure call, execute the call, pack the results and finally transfer the results. You will get the best performance if you design your remote procedures with this in mind. Remote procedures that can gather as much information as possible in one call, will perform more efficiently than procedures that return a single item of information and must be called a number of times. Consider the PRIMOS subroutines DIR$RD and DIR$LS. Both these procedures return directory entries and may be called remotely, however, using DIR$LS is the better "remote" choice. This is so because DIR$LS can return multiple directory entries in one call. For example, you can obtain 30 directory entries and only pay the NPX overhead once. If you used DIR$RD, you could spend more time in NPX than you would in DIR$RD.

5 Problems

NPX is under the control of PRIMOS, not ICE. The most obvious problem is releasing your slave when your application terminates abnormally. ICE attempts to correct this situation by using a static onunit. When "terminal" conditions are raised, all the user's slaves will be released. There will be situations where this is not possible. At this time, the only method of cleanup is to logout of PRIMOS.

6 DOPE.INS - Dope Vector Descriptor

FILE: <MSP194>SYSLIBSRC>INSERT>DOPE.INS.SPL

DESCRIPTION:

This file contains the replacements and description of the compiler generated dope vectors for parameters with "*" or "variable" extents. Callers of ICE_PCL must supply dope vector descriptors any time a remote procedure has at least one parameter declared using a variable extent. This is true for arrays of any type, and character strings.

After the last declared parameter is passed, you will begin to pass dope vector descriptors. The vectors are positional and correspond identically to the calling sequence of the declared parameters. A dope vector must be passed for each parameter in the list. If a parameter does not have a variable extent, the value DV$FILL may be passed. This value is passed to "fill" the space between parameters that have variable extents, the compilers do not examine it.

Although you must "fill in the gaps", it is not necessary to "fill" the entire parameter list with DV$FILL once you have satisfied the requirement for the last variable extent parameter. This is better shown in the example below. You want to call the following remote procedure:

```
print:
   proc(mbz1, string, userid, mbz2);

   dcl mbz1 fixed bin;
   dcl string char(*);
   dcl userid char(32) var;
   dcl mbz2 fixed bin;

      put skip list('Message from', userid);
      put skip list(string);
      return;

end print;
```

Your procedure would be written as follows:

```
   main:
      proc;

      dcl error fixed bin;
      dcl slave_p pointer;
      dcl string char(32);
      dcl mbz1 fixed bin;
      dcl mbz2 fixed bin;
      dcl string_dv like dope_vector;
      dcl userid char(32) var;
      dcl ice_pcl entry options(variable);

      ...

         string_dv.type = dv$char;
         string_dv.ndims = '0'b;
         string_dv.size = 32;

         string = 'The slave will print this string.';
         userid = 'ME';

         call ice_pcl(ik$pcl, slave_p, 'PRINT', 5, error,
            mbz1, 1, binary(ik$fb15 + ik$in, 15),
            string, 32, binary(ik$char + ik$in, 15),
            userid, 32, binary(ik$vchr + ik$in, 15),
            mbz2, 1, binary(ik$fb15 + ik$in, 15),
            dv$fill, 1, binary(ik$fb15 + ik$in, 15),
            string_dv, 2, binary(ik$fb15 + ik$in, 15));

      ...
   end main;
```

Please note the use of DV$FILL in the above example. It was required to insure that the descriptor for "string" was the fourth parameter in the list. Since "userid" does not have a variable extent and there are no more variable extent parameters in the list, DV$FILL does not have to be supplied.

ABNORMAL-CONDITIONS:

If you fail to supply these dope vectors, the condition "POINTER_FAULT$" will be raised in the slave process and the procedure call will fail.

If you are not sure how the called procedure has declared its parameters, passing a dope vector will always work, even if the parameters do not have "*" extents.

DECLARATION:

```
   dcl 1 dope_vector based,
         2 type bit(8),
         2 ndims bit(8),
         2 size fixed bin,       /* Depends on data type:
                                    arithmetic  declared Q*256+P
                                    string      declarled length
                                    pictured    address of edit sub
                                    area        size of area
                                    otherwise the field is zero */

         2 bound(8),                /* only needed for arrays */
            3 lower fixed bin(31),  /* lower bound */
            3 upper fixed bin(31),  /* upper bound */
            3 span fixed bin(31);   /* distance between elements */


%replace dv$fill by 'FFFF'b4;           /* Filler */
%replace dv$pictured by '01'b4;         /* PICTURED */
%replace dv$fixedbin by '02'b4;         /* FIXED BINARY */
%replace dv$floatbin by '03'b4;         /* FLOAT BINARY */
%replace dv$fixeddec by '04'b4;         /* FIXED DECIMAL */
%replace dv$floatdec by '05'b4;         /* FLOAT DECIMAL */
%replace dv$comfixbin by '06'b4;        /* COMPLEX FIXED BINARY */
%replace dv$comfltbin by '07'b4;        /* COMPLEX FLOAT BINARY */
%replace dv$comfixdec by '08'b4;        /* COMPLEX FIXED DECIMAL */
%replace dv$comfltdec by '09'b4;        /* COMPLEX FLOAT DECIMAL */
%replace dv$char by '0A'b4;             /* CHARACTER */
%replace dv$charvar by '0B'b4;          /* CHARACTER VARYING */
%replace dv$bit by '0C'b4;              /* BIT */
%replace dv$bitvar by '0D'b4;           /* BIT VARYING */
%replace dv$bitalign by '0E'b4;         /* BIT ALIGNED */
%replace dv$pointer by '0F'b4;          /* POINTER */
%replace dv$offset by '10'b4;           /* OFFSET */
%replace dv$area by '11'b4;             /* AREA */
%replace dv$file by '12'b4;             /* FILE */
```

```
%replace dv$label by '13'b4;          /* LABEL */
%replace dv$entry by '14'b4;          /* ENTRY */
%replace dv$logical by '15'b4;        /* FTN LOGICAL */
```

7 ICE ALOC - Allocate Slave Process

FILE: <MSP194>SYSLIBSRC>ICESRC>ICE_ALOC.SPL

DESCRIPTION:

ICE_ALOC allocates a slave process on the specified node. The virtual circuit between the local node and target node is established when it called for the first time. ICE_RLS must be called to release the slave.

USAGE:

```
dcl ICE_ALOC entry(fixed bin, char(32) var, pointer, fixed bin)
             returns(bit(1));

success = ICE_ALOC(key, nodename, slave_p, error);
```

key	Possible values are: IK$ANY If any slave has been started on this node, increment the allocation count of the first slave that was allocated and return the slave info pointer. If no slaves have been started, allocate one and return the information pointer. IK$USE Use the SLAVE_P argument and increment the allocation count for this slave.
nodename	The ASCII name of the target node. Not used if the key is IK$USE. Leading and/or trailing blanks are ignored, and case does not matter. Type: char(32) var (input parameter)
slave_p	A pointer to information about the slave

	process being allocated. Usage of this parameter is dependent upon the supplied key value. The caller must submit this value in subsequent calls to ICE_ALOC, ICE_PCL and ICE_RLS. Type: pointer (input/output parameter)
error	Results. Possible values are:

0	Operation complete.
E$MSLV	Maximum number of slaves allowed per user has been exceeded.
E$NETE	Network Error
E$RLDN	Remote Line is Down
E$NSLA	No slaves available
E$BPAR	Parameters are invalid
E$RSNU	Remote system not up

Type: fixed bin (output parameter)

ABNORMAL-CONDITIONS:

None.

8 ICE KEYS.INS - Mnemonic Keys For ICE

FILE: <MSP194>SYSLIBSRC>INSERT>ICE_KEYS.INS.SPL

DESCRIPTION:

Mnemonic keys used for ICE procedure calls.

ABNORMAL-CONDITIONS:

None.

DECLARATION:

```
************************ ICE_ALOC ************************

%replace ik$any by 1,                  /* Any slave or new one */
         ik$new by 2,                  /* Allocate new slave */
         ik$use by 3;                  /* Use specific slave */

************************  ICE_RLS  ************************

%replace ik$all by 4,                  /* Release ALL Slaves */
         ik$spec by 5;                 /* Release specific slave */

************************  ICE_NAME ************************

%replace ik$mine by 6,                 /* Return local node name */
         ik$slave by 7;                /* Return node of slave */

************************  ICE_PCL  ************************

%replace                               /* Call Type Keys */
         ik$pcl by 0,                  /* It's a procedure call */
         ik$func by 8192,              /* It's a function, return L-REG
                                          */
         ik$rtry by 16384;             /* Retry if slave not available
                                          */
```

```
%replace                                   /* Argument Type Keys */
         ik$fb15 by 0,                     /* argument is fixed bin */
         ik$i2 by 0,

         ik$fb31 by 256,                   /* argument is fixed bin(31) */
         ik$i4 by 256,

         ik$char by 512,                   /* argument is character */

         ik$vchr by 768,                   /* argument is character varying
                                              */
         ik$ptr by 1024,                   /* argument is a pointer */

         ik$fl23 by 1280,                  /* argument is float bin(23) */
         ik$r4 by 1280,

         ik$fl47 by 1536,                  /* argument in float bin(47) */
         ik$r8 by 1536;

%replace ik$in by 128,                     /* argument is INPUT */
         ik$out by 64,                     /* argument is OUTPUT */
         ik$ref by 0;                      /* argument is a reference */
```

9 ICE KEYS.INS - Mnemonic Keys For ICE

FILE: <MSP194>SYSLIBSRC>INSERT>ICE_KEYS.INS.F77

DESCRIPTION:

Mnemonic keys used for ICE procedure calls.

ABNORMAL-CONDITIONS:

None.

DECLARATION:

```
      INTEGER*2 IK$ANY, IK$NEW, IK$USE, IK$ALL, IK$SPEC,
     C IK$MINE, IK$SLAVE,
     C IK$PCL, IK$FUNC, IK$RTRY, IK$I2, IK$I4, IK$CHAR,
     C IK$VCHR, IK$LOC, IK$R4, IK$R8, IK$IN, IK$OUT,
     C IK$REF

 ************************ ICE_ALOC ************************

      PARAMETER IK$ANY = 1              /* Any slave or new one */
      PARAMETER IK$NEW = 2              /* Allocate new slave */
      PARAMETER IK$USE = 3              /* Use specific slave */


 ************************  ICE_RLS ************************

      PARAMETER IK$ALL = 4              /* Release ALL Slaves */
      PARAMETER IK$SPEC = 5             /* Release specific slave */


 ************************  ICE_NAME ************************

      PARAMETER IK$MINE = 6            /* Return local name node
      PARAMETER IK$SLAVE = 7           /* Return node of slave


 ************************  ICE_PCL ************************
```

```
      PARAMETER IK$PCL = 0                /* It's a procedure call */
      PARAMETER IK$FUNC = 8192            /* It's a function,
                                             return L-REG */
      PARAMETER IK$RTRY = 16384           /* Retry if slave not
                                             available */

      PARAMETER IK$I2 = 0                 /* argument is INTEGER*2 */
      PARAMETER IK$I4 = 256               /* argument is INTEGER*4 */
      PARAMETER IK$CHAR = 512             /* argument is CHARACTER */
      PARAMETER IK$VCHR = 768             /* argument is CHAR VAR */
      PARAMETER IK$LOC = 1024             /* argument is a LOC */
      PARAMETER IK$R4 = 1280              /* argument is REAL*4 */
      PARAMETER IK$R8 = 1536              /* argument in REAL*8 */

      PARAMETER IK$IN = 128               /* argument is INPUT */
      PARAMETER IK$OUT = 64               /* argument is OUTPUT */
      PARAMETER IK$REF = 0                /* argument is a reference */
```

10 ICE NAME - Return Node Name

FILE: <MSP194>SYSLIBSRC>ICESRC>ICE_NAME.SPL

DESCRIPTION:

ICE_NAME is used to return the node name of the local node, or the node on which a specific slave has been allocated.

USAGE:

```
dcl ICE_NAME entry(fixed bin, pointer, char(32) var, fixed bin)
        returns(bit(1));

success = ICE_NAME(key, slave_p, name, error);
```

key	May be one of the following: IK$MINE Return local node name IK$SLAVE Return node of slave Type: fixed bin (input parameter)
slave_p	The information pointer that identifies the slave. The node on which this slave has been allocated will be returned. This parameter is used only if the key is IK$SLAVE. Type: pointer (input parameter)
name	The returned node name. Will be set to the null string if networks are not configured and the key is IK$MINE. Type: char(32) var (output parameter)
error	Results. Possible values are: 0 Success completion. E$PTRM The Slave information pointer

	is invalid.
E$BPAR	The key is invalid.

ABNORMAL-CONDITIONS:

None.

11 ICE PCL - Execute Remote Procedure Call

FILE: <MSP194>SYSLIBSRC>ICESRC>ICE_PCL.SPL

DESCRIPTION:

This subroutine is the ICE interface to execute remote procedure calls. It passes the subroutine name and arguments to a previously allocated slave. The slave process then calls the specified procedure on the remote system. The procedure in question must be dynamically linkable, i.e, in a shared library or PRIMOS. Note that each argument to the local subroutine expands to a triplet of arguments to this subroutine. Please note that a maximum of 4K words may be transfered in one argument.

USAGE:

```
dcl ICE_PCL entry(fixed bin, pointer, char(32), fixed bin, fixed bin,
                     [variable]) [returns fixed bin or fixed bin(31)];

[function =] or [call]
   ICE_PCL(key, slave_p, procname, proclen, error,
            arg1, arg1len, arg1type, ..., argn, argnlen, argntype);
```

key	May be one of the following: IK$PCL — This call is a procedure call. IK$FUNC — This call is a function, the L-REG is returned. Type: fixed bin (input parameter)
slave_p	Pointer to the slave information as returned by ICE_ALOC. Type: pointer (input parameter)
procname	The ASCII name of the procedure to call. Leading and trailing blanks are ignored and case does not matter. According to the current search rules, this procedure must be dynamically linkable.

	Type: char(32) (input parameter)
proclen	The number of non-blank characters in procname. Type: fixed bin (input parameter)
error	The results of the remote call attempt. This parameter is NOT passed to the remote procedure. Possible values are:

0	Call was successfully transmitted and has been executed.
E$BPAR	The caller's arguments to this procedure are invalid.
E$PNTF	Remote Procedure Not Found.
E$BCFG	Network configuration mismatched between nodes.
E$VCGC	The virtual circuit got cleared.
E$RLDN	The remote line is down
E$NSLA	No Slaves Available
E$RSNU	The remote system is not up yet.
E$MNPX	Multiple hops in NPX. Slaves cannot allocate slaves.
E$NBUF	No buffer space. Argn length is > 4K words.

	Type: fixed bin (output parameter)
argN	The Nth argument to the target subroutine. A maximum of 15 arguments are supported. All "argN" arguments are optional, they do not have to be supplied. However, for each "argN" that is supplied a corresponding "argNlen" and "argNtype" pair of arguments

must be supplied.

Type: any type (any direction)

argNlen	The length of the Nth argument. This length is represented in its basic unit as identified by the argNtype (see below). May not exceed 4K words of data in either direction. Type: fixed bin (input parameter)

argNtype

An additive key that identifies the type of the argument being passed.

May be one of the following:

IK$FB15 or IK$I2
argN is a FIXED BIN(15) whose basic length unit is 1 16 bit word.

IK$FB31 or IK$I4
argN is a FIXED BIN(31) whose basic length unit is 2 16 bit words.

IK$CHAR
argN is a character string whose basic length unit is 8 bits.

IK$VCHR
argN is a PL/1 character varying string. whose basic length unit is 8 bits. Do add an extra 2 characters for the length word, it will be taken into consideration.

IK$PTR or IK$LOC
argN is an address whose basic length unit is 1 16 bit word.

*** Note ***
The length must represent the number of 16 bit words the pointer addresses NOT the size of the pointer itself.

IK$FL23 or IK$R4
argN is a FLOAT BIN(23) whose basic length unit is 4 16 bit words.

IK$FL47 or IK$R8
argN is a FLOAT BIN(47) whose basic length unit is 8 16 bit words.

Plus:

IK$IN	ArgN is an input argument. Input arguments are SENT to the slave. They are NOT passed back.

Plus:

IK$OUT	argN is an output argument. Output arguments are NOT sent to the slave, they are received.

Type: fixed bin (input parameter)

ABNORMAL-CONDITIONS:

The link may go down during this call or between calls (the slave is kept in waiting between successive calls); ICE tries to recover from this failure, but in the event of an unsuccessful recovery, the error can be reported to the user via the ERROR argument (see the E$XXXX labels for other error conditions).

If the procedure to be called has any parameters declared with "*" extents, ICE_PCL will not function properly. The compilers generate a dope vector for each parameter of this type. At this time, ICE will not generate these dope vectors. You may pass these dope vectors as arguments to ICE_PCL. An insert file is supplied which describes these vectors, see DOPE.INS. later on in this manual. This insert file will be supplied in the directory SYSCOM.

If you declare ICE_PCL as entry options(variable), you must pass additive keys using the binary built-in function. For example:

```
binary(ik$fb15 + ik$in + ik$out, 15)
```

At this time it is known why the compiler does not produce the correct result for additive keys. Using the binary function will always product the desired result.

12 ICE RLS - Release An ICE Slave

FILE: <MSP194>SYSLIBSRC>ICESRC>ICE_RLS.SPL

DESCRIPTION:

This procedure performs a specified number of releases on a slave. At any time when the count of allocations becomes zero, the connection between the slave and the master is broken, allowing the slave to be freed from the caller. If the resulting count of allocations is not zero, the connection is kept open.

USAGE:

```
dcl ICE_RLS entry(fixed bin, pointer, fixed bin, fixed bin)
            returns(bit(1));

success = ICE_RLS(key, slave_p, count, error);
```

key	Possible values are: IK$ALL — Release all slaves for all allocations. IK$SPEC — Release the slave as identified by SLAVE_P Type: fixed bin (input parameter)
slave_p	The pointer to the slave information that must be released. Not used for IK$ALL. Type: pointer (input/output parameter)
count	The number of releases to perform. If count is zero, release all allocations and break the virtual circuit between the slave and the master. Otherwise, perform "count" releases. Not used for IK$ALL.

	Type: fixed bin (input parameter)
error	Results. Possible values are:

0	Operation complete.
E$PTRM	Pointer Mismatch. The slave information pointer is not valid.
E$BPAR	The count of allocations on this slave is less than the number of release to perform.
E$BVVC	Problems in clearing the virtual circuit.
E$VCGC	The virtual circuit got cleared before the slave could be released.

Type: fixed bin (output parameter)

<u>ABNORMAL-CONDITIONS:</u>

None.