Using a Conversational RPC

This section covers the following topics:

- Opening a Conversation
- Closing a Conversation
- Defining a Conversation Context
- Modifying the System Variable *CONVID

Opening a Conversation

To open a conversation

1. Specify an `OPEN CONVERSATION` statement on the client side.
2. In the `OPEN CONVERSATION` statement, specify a list of services (subprograms) as members of this conversation.

The `OPEN CONVERSATION` statement assigns a unique conversation identifier to the system variable `*CONVID`.

More than one conversation may be open in parallel. If subprograms interfere with each other, the application programs are responsible to manage the various conversations by setting the appropriate `*CONVID`, which is evaluated by the `CALLNAT` instruction.

- If the subprogram is a member of the current conversation (referred to by `*CONVID`), it will be executed at the server task which is exclusively reserved for this conversation.
- If it is not member of the current conversation, it will be executed in a different server task. This also applies to different conversations.

A conversation can be opened on any program level and `CALLNAT`s within this conversation can be executed on any other program level below or above.

It is possible to open a client conversation within a remote `CALLNAT` executed on a server so the server acts as an agent. As the client only controls its own conversations, and not the server's, it is the application programmer's responsibility to ensure that the conversation on the server is closed properly before the main client is closed.

Additional Restrictions

The conversational RPC can still be tested locally. To keep the behavior identical if you execute a conversational `CALLNAT` remotely or locally, the following additional restrictions apply:

- A `CLOSE CONVERSATION` is not possible within an object which is currently running as a member of this conversation. This corresponds to the restriction that it is not possible to close a conversation from within a remotely running program.

- It is not possible to execute a conversational CALLNAT which is member of the conversation from within another (or the same) member of this conversation. This corresponds to the restriction that it is not possible to execute a conversational CALLNAT which is member of the client's conversation from a server subprogram.
- It is not recommended to open a conversation from within another conversation's subprogram.

Closing a Conversation

To close a conversation

- Specify a CLOSE CONVERSATION statement on the client side.

This enables the client to close a specific conversation or all conversations. All context variables of the closed conversation are then released and the server task will be available again for another client.

If you terminate Natural, you implicitly close all conversations.

When a server receives a CLOSE CONVERSATION request, it issues a CLOSE CONVERSATION ALL statement so that all conversations the server might have opened (as agent) are also closed.

Close a conversation with implicit BACKOUT TRANSACTION (Rollback)

By default, when a CLOSE CONVERSATION statement is executed, the Rollback option will be sent to the server together with the CLOSE CONVERSATION statement. This will cause an implicit BACKOUT TRANSACTION on the server side at the end of the conversation processing.

Close a conversation with implicit END TRANSACTION (Commit)

You can use the application programming interface USR2032N available in library SYSEXT to cause an implicit END TRANSACTION on the server side.

The application programming interface has to be called before the next CLOSE CONVERSATION statement is executed. The result is that the commit option is sent to the server together with the CLOSE CONVERSATION statement and that the server executes an END TRANSACTION statement at the end of the conversation processing.

The commit option applies to the next CLOSE CONVERSATION statement executed by the client application. After the conversation(s) has (have) been closed, the default option is used again. This means, that the following CLOSE CONVERSATION statements will result again in a BACKOUT TRANSACTION statement.

Defining a Conversation Context

During a conversation the subprograms that are members of this conversation may share a context area on this server.

To do so, declare a data area with the DEFINE DATA CONTEXT statement in each of the concerned subprograms.

The subprograms, using a context area, behave in the same way if the conversation were local or remote. The DEFINE DATA CONTEXT statement closely corresponds to the DEFINE DATA INDEPENDENT statement. All rules which apply to the definition of AIV variables also apply to context variables, with the exception that a context variable does not need to be prefixed by a plus sign (+).

The compiler does not check format/length definition because this requires that the variables be created by running a program which includes all definitions for this application (as usual with AIVs). This makes no sense for context variables, because a library containing RPC service routines is usually not application-dependent.

In contrast to AIVs, the caller's context variables are not passed across CALLNAT boundaries. Context variables are referenced by their name and the context ID they apply to. A context variable is shared by all service routines referring to the same variable name within one conversation. Therefore each conversation has its own set of context variables. Context variables cannot be shared between different conversations even if they have the same variable name.

The context area will be reset to initial values when an OPEN CONVERSATION statement or a non-conversational CALLNAT statement is performed.

Modifying the System Variable *CONVID

The system variable *CONVID (format I4) is set by the OPEN CONVERSATION statement and may be modified by the application program.

Modifying *CONVID is only necessary if you are using multiple conversations in parallel.