**MSCRCHAN - Channels and Containers**

- Introduction
- HBX Receiving Control with a Channel
- HBX LINKing to a Program with a Channel
- MSCRCHAN methods
  - deleteContainer
  - endBrowse
  - getContainer
  - getName
  - getNext
  - getRC
  - getResp
  - getResp2
  - moveContainer
  - putContainer
  - setName
  - startBrowse
- Example
  - Sample Script "mychan"
  - Sample Script "mysub"
  - Sample Output

**Introduction**

The IBM documentation for CICS TS Version 3 provides the following overview of channels and containers:

"Traditionally, CICS application programs have used communication areas (COMMAREAs) to exchange data. However, CICS TSVersion 3.1 provides an improved method of transferring data between programs, in amounts that far exceed the 32KB limit that applies to COMMAREAs. Containers are named blocks of data designed for passing information between programs. You can think of them as "named communication areas (COMMAREAs)". Programs can pass any number of containers between each other. Containers are grouped together in sets called channels. A channel is analogous to a parameter list.

To create named containers and assign them to a channel, a program uses EXEC CICS PUT CONTAINER(container-name) CHANNEL(channel-name) commands. It can then pass the channel (and its containers) to a second program using the CHANNEL(channel-name) option of the EXEC CICS LINK, XCTL, START, or RETURN commands.

The second program can read containers passed to it using the EXEC CICS GET CONTAINER(container-name) command. This command reads the named container belonging to the channel that the program was invoked with.

If the second program is invoked by a LINK command, it can also return containers to the calling program. It can do this by creating new containers, or by reusing existing containers. Channels and containers are visible only to the program that creates them and the programs they are passed to. When these programs terminate, CICS automatically destroys the containers and their storage."

The MSCRCHAN object provides an interface to the CICS TS 3.1 services, and the EXEC CICS commands, that support channels and containers.

Channels and containers come into play within the HBX environment in two ways. First, the HBX engine can receive control with input data specified using a channel. Secondly, HBX can LINK to another program and pass data to it as containers within a channel. In either case, an HBX script accesses the channel, and the containers within it, using the MSCRCHAN object.

**HBX Receiving Control with a Channel**

HBX supports a number of different transport mechanisms for receiving requests and returning responses (HTTP, MQ, DPL, etc.). Upon receiving control, HBX must determine the name of the script to be executed. The first step in this process is for HBX to inspect its execution environment and determine if it has received its input via a commarea or a channel.

If HBX receives control with a channel, it then evaluates the name of the channel. The name of the channel can infer information regarding the underlying transport, and the type of request to be processed. HBX recognizes the following reserved channel names:

DFHAHC-V1
HB-NODE

In these cases, HBX will inspect the containers within the channel to derive the name of the script to be executed and other processing parameters. Otherwise, HBX will examine the channel to see if a container named HBXscript is included. If so, HBX expects the content of this container to be a string which specifies the name of the script to be executed. Otherwise, HBX will assume that the name of the channel is the name of the script. In either case, HBX will pass the names of the containers in the channel as arguments to the script. Furthermore, HBX will automatically redirect the output from write and writeln statements within the script to a container named HBXoutput. This allows the script to function with a high degree of isolation relative to the transport.

**HBX LINKing to a Program with a Channel**

A channel, and containers within it, can also be created by an HBX script and passed to another program. To facilitate this, the link method of the MSCRICICS object supports the specification of either a commarea or a channel as the second argument.

**MSCRCHAN methods**
The MSCRCHAN methods are:

- deleteContainer – Deletes a container from the current channel.
- endBrowse – Ends a browse operation initiated with the startBrowse method.
- getContainer – Returns the contents of a container in the current channel as a variable.
- getName – Returns the name of the current channel.
- getNext – Returns the name of the next container in the current channel (used in conjunction with startBrowse and endBrowse methods).
- getRC – Gets the last CICS RESP value.
- getResp – Gets the last CICS RESP value.
- getResp2 – Gets the last CICS RESP2 value.
- moveContainer – Moves a container, and its contents, from the current channel to another. Can also be used to rename a container in the current channel.
- putContainer – Puts containers and values in a channel and displays the return code.
- setName – Sets name for a channel.
- startBrowse – Starts loop through container names and displays return code.

Unless otherwise specified, the following examples assume that the MSCRCHAN object has been defined as follows:

HostLib(chan,"MSCRCHAN");

In the following method descriptions, the "current channel" refers to either the channel received by HBX, or the channel specified on the most recent setName method call.

**deleteContainer**

*Description:* Deletes a container from the current channel. Returns the RESP value from the EXEC CICS DELETE CONTAINER command.

*Syntax:*

```plaintext
deleteContainer(containerName);
```

*Parameters:*

- `containerName` - Specifies the name of the container to delete.

*Returned value:* Numeric

*Example:* `writeln("DEL=",chan.deleteContainer("MYCONT3"));`

**endBrowse**

*Description:* Ends a browse operation initiated with the startBrowse method. Returns the RESP value from the EXEC CICS ENDBROWSE CONTAINER command.

*Syntax:*

```plaintext
drop()
```

*Parameters:* None.

*Returned value:* Numeric

*Example:* `chan.endBrowse();`

**getContainer**

*Description:* Returns the contents of a container.

*Syntax:*

```plaintext
getContainer(containerName,chan.type,size,[#decimals]);
```

*Parameters:*

- `containerName` - Specifies the name of the container from which to retrieve contents.
- `chan.type` - Specifies the data type of the container contents (e.g., "string").
- `size` - Specifies the length of a decimal value retrieved from a container.
#decimals - Specifies the number of decimal points of a decimal value retrieved from a container.

** Returned value:** A variable of the specified type.

** Example:** writeln("GET=",chan.getContainer("HBXoutput",chan.string));

### getName

**Description:** Returns the name of the current channel.

**Syntax:**

getName();

**Parameters:** None

** Returned value:** 1 to 16 character string

** Example:** cname = chan.getName();

### getNext

**Description:** Returns the name of the next container in the current channel. Used in conjunction with the startBrowse/endBrowse methods to obtain the names of the containers in the current channel.

**Syntax:**

getNext();

**Parameters:** None

** Returned value:** 1 to 16 character string with container name

** Example:** writeln("NXT=",chan.getNext);

### getRC

**Description:** Gets the RESP value returned from the last EXEC CICS command executed by the object. Synonymous with the getResp method.

**Syntax:**

getRC();

**Parameters:** None

** Returned value:** Numeric

** Example:** rc = chan.getRC();

### getResp

**Description:** Gets the RESP value returned from the last EXEC CICS command executed by the object. Synonymous with the getRC method. This method is can be used to get response information following the execution of a method which returns a value other than the response information (e.g. getName).

**Syntax:**

getResp();

**Parameters:** None

** Returned value:** Numeric

** Example:**
```javascript
while(true) {
    a = chan.getNext(); if(chan.getResp() != 0) break; writeln("NXT=",a);
```
getResp2

**Description:** Gets the RESP2 returned from the last EXEC CICS command executed by the object.

**Syntax:**
getResp2();

**Parameters:** None

**Returned value:** Numeric

**Example:** writeln("RESP2=".chan.getResp2());

moveContainer

**Description:** Moves a container, and its contents, from the current channel to another. If the name of a channel is not specified (using the third parameter), the effect of this method is to rename a container which the current channel. Returns the RESP value from the EXEC CICS MOVE CONTAINER command.

**Syntax:**
moveContainer(containerName,newContainerName [toChannel]);

**Parameters:**
containerName - Specifies the name of the container to be moved.
newContainerName - Specifies the new name for the specified container.
toChannel - Specifies the name of the channel to which the container is to be moved. If not specified, the current channel will be used.

**Returned value:** Numeric

**Example:**
// move a container to a new name, it becomes MYCONT3
writeln("MOV=".chan.moveContainer("CONT","CONT3"));

putContainer

**Description:** Creates a container by the specified name within the current channel, and stores the specified data in the container. Returns the RESP value from the EXEC CICS PUT CONTAINER command.

**Syntax:**
putContainer(containerName,containerData);

**Parameters:**
containerName - Specifies the name of the container to be created.
c
containerData - Places data into the specified container.

**Returned value:** Numeric

**Example:** writeln("PUT=".chan.putContainer("CONT1","Hello World"));

setName

**Description:** Sets a text handler. This handler receives a text string that is built from the data between a start and end tag.

**Syntax:**
setName(channelName);
**Parameters:**

`containerName` - The name of the current channel to be used for all subsequent method calls.

*Returned value:* Numeric

*Example:* `chan.setName("MYCHAN");` // give our channel a name

---

### startBrowse

**Description:** Initiates browsing of the active channel. The names of the containers in the channel are retrieved with the getNext method. Returns the RESP value returned by the EXEC CICS STARTBROWSE CONTAINER command.

**Syntax:**

`startBrowse(channelName);`

**Parameters:**

`channelName` - Specifies the name of the channel to be browsed. If this parameter is not specified, the current channel will be used.

*Returned value:* Numeric

*Example:* `chan.startBrowse("CONT3");`

---

**Example**

The following example illustrates how one script can invoke another script and pass it data within a channel. The first script, "mychan", creates a channel, containers within it, and LINKs to a program. The program LINKed to is MSCRIPT, which illustrates how HBX interprets the content of the HBXscript container as the script name ("mysub"). Mysub uses write and writeln statements to generate its output, which will be returned to mychan in the HBXoutput container.

To run this sample, perform a Make operation on mysub first (so the HBX can find it when MSCRIPT is LINKed to). Then, you canRun mychan.

**Sample Script “mychan”**
HostLib(cics,"MSCRCICS",chan,'MSCRCHAN');

// This script illustrates how a channel can be created and
// passed to program via LINK.
// In this case, the LINKed to program is also a script.

function mychan()
  writeln("mychan beginning");
  chan.setName("MYCHAN"); // give our channel a name
  // put the name of the script to be invoked in container HBXscript
  writeln("PUT=",chan.putContainer("HBXscript","mysub"))); // create 2 other containers with some sample data
  writeln("PUT=",chan.putContainer("MYCONT","Hello World");
  writeln("PUT=",chan.putContainer("MYCONT2","Hello World Again");

  // browse the containers we just created
  writeln("BEG=");while(true)
  {
    a = chan.getNext();
    if (chan.getResp() != 0) break;
    writeln("NXT=",a);
  }
  writeln("END=");

  // get the contents of a container
  writeln("GET=");chan.getContainer("MYCONT",chan.string);

  // move a container to a new name in the same channel
  // (i.e., rename the container)
  writeln("MOV=");chan.moveContainer("MYCONT","MYCONT3");

  // delete the container we just created
  writeln("DEL=");chan.deleteContainer("MYCONT3");

  // invoke the HBX engine and pass it the channel
  writeln("linking to mysub");
  rc = cics.link("MSCRIPT", chan);
  writeln("returned from mysub");
  writeln("The following output was generated by mysub:","\n");
  // display the output from the script
  writeln(chan.getContainer("HBXoutput",chan.string));
  writeln("mychan ending");
}
mychan();

Sample Script “mysub”
HostLib(cics,"MSCRCICS",chan,'MSCRCHAN');

function mysub()

// This script assumes it was invoked by a channel.
// It will return information about the containers it received. writeln("mysub beginning");

// The names of the containers in the channel will be received as
// arguments to the script. Echo them back to the calling script. writeln("mysub received the following containers in channel:",
chan.getName());

for (i=0;i<arguments.length;i++)
{
    writeln(arguments[i],i+1);
}
writeln("The following string was in container MYCONT2:"); writeln("GET=",chan.getContainer("MYCONT2",chan.
    string)); writeln("mysub ending");
}

Sample Output

mychan beginning
PUT= 0
PUT= 0
PUT= 0
BEG= 0
NXT=MYCONT2
NXT=MYCONT NXT=HBXscript END= 0
GET=Hello World
MOV= 0
DEL= 0

linking to mysub

returned from mysub

The following output was generated by mysub:

mysub beginning

mysub received the following containers in channel:MYCHAN

MYCONT2
HBXscript

The following string was in container MYCONT2: GET=Hello World Again

mysub ending

mychan ending