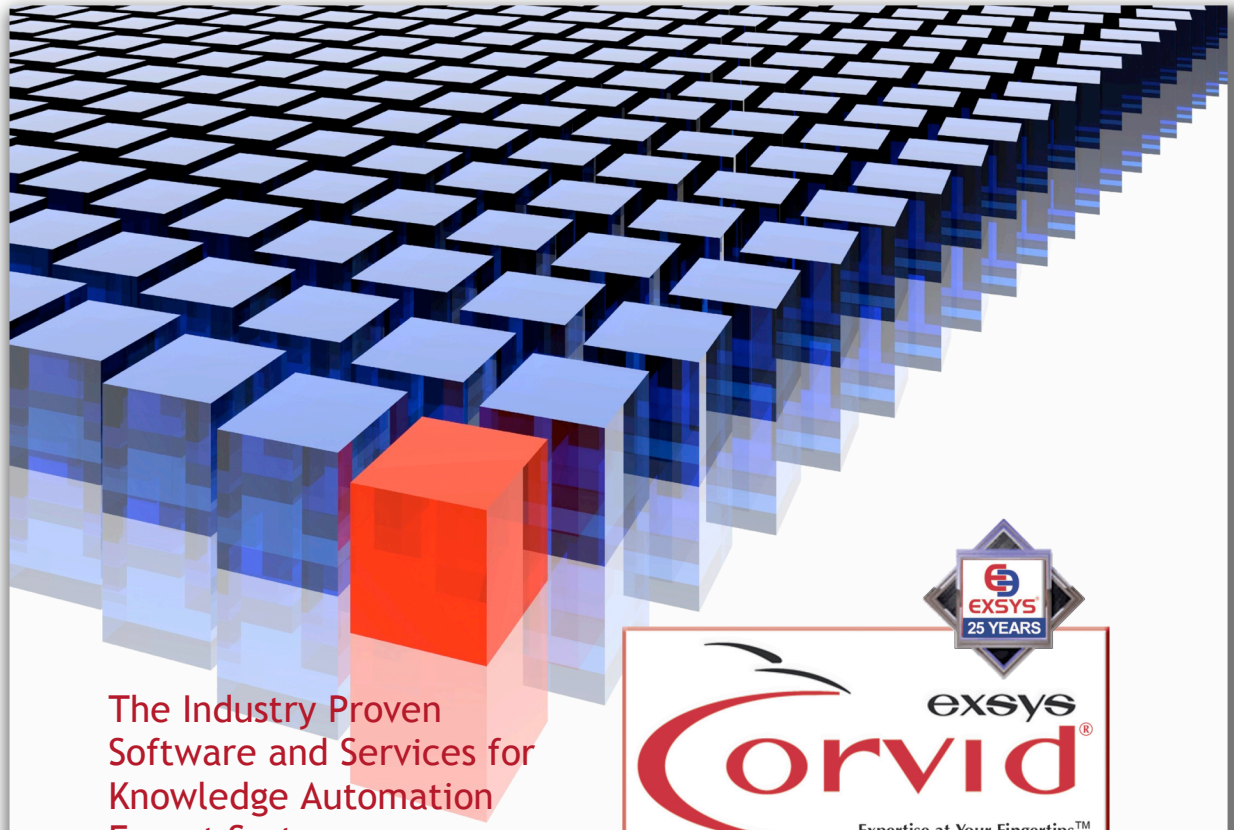


The Industry Proven Software and Services for Knowledge Automation Expert Systems

exsys Corvid®

Expertise at Your Fingertips™

Exsys Corvid® v5.0 New Features Addendum

Integrating Exsys Corvid with Adobe Flash

Exsys Corvid v5 enables Exsys developers to use Adobe Flash to design the user interface for Exsys knowledge automation systems. This allows the tremendous design, animation and multimedia capabilities of Flash to be used to ask questions, present results and display information. Flash is the premiere multimedia tool for making web sites "pretty", now you can make them pretty and smart. Exsys Corvid systems can interact with the end user as if they were talking to a human expert, with all the Flash capabilities for graphic design and data presentation. Flash applications can now use the Exsys Inference Engine to perform complex analysis or drive interactive sessions that would not be practical to "hard code" into a system.

Using Flash for the user interface greatly expands the delivery options beyond the previous Exsys Runtime programs, which while more than adequate for many types of interactive expert systems, lack the ability to do complex graphics, animation or multimedia. Flash based systems can make use of the full design capabilities of the Adobe Flash IDEs - either Adobe Flash CS4 or Adobe Flex. These powerful, proven and integrated tools are widely used to produce the core part of many web sites. There are many books on using Flash and an extensive user community.

The underlying Flash programming language, Action Script 3, makes it easy to connect a Flash application to Exsys. Action Script 3 also is a full and robust language that can be used in conjunction with Exsys to add special interfaces or program custom functionality into a system. This can allow some aspects of the system such as dynamic questions and input validation to be handled client-side in Flash, reducing the load of the server-side computation.

An additional benefit is that the Flash SWF file can run in a small window on a page, similar to an applet window, rather than having the Exsys system fully rebuild an entire page. This allows integrating an expert system with a complex interface into a larger web page design, while still allowing it to run independent of the rest of the page. The end user machine need only have the standard Adobe Flash player, which is generally included with most browsers, rather than a Java plug-in.

Using Flash for the interface requires having the Adobe Flash development tools and knowledge of how to build Flash applications. While commonly available, these programs are not provided by Exsys, and the Exsys/Flash documentation only covers how to connect Exsys systems with Flash - not how to use the Flash IDEs themselves. If you already are familiar with Flash, you're all set. If not, the Adobe Flash tutorials and many introductory books on Flash should get you going quickly. Flash is an interesting and fun program to learn since it makes it easy to do things that would be very difficult to do otherwise. The Exsys/Flash interface is based on Action Script 3 commands. You will need to know some Action Script to build your specific system, but the core code to interface Exsys and Flash is provided and can just be pasted into your system.

To make use of Flash, you will also need the Exsys Corvid Servlet Runtime v5. This version can be used to run Exsys systems with Flash SWF files, or (as in earlier versions) build standard HTML forms and pages. If you are already running systems with the Servlet Runtime, they will continue to run as before, however, the Servlet Runtime v5 adds the ability to run with Flash SWF applications. You can mix Flash and non-Flash systems on the same Corvid Servlet Runtime.

The rules and logic in the Exsys knowledge automation systems you build are the same for Flash and non-Flash systems. Existing Exsys systems can have a Flash user interface added to them without changing the core logic - though new systems may be designed to make use of the special Flash capabilities that were not previously available.

Using Flash for the User Interface

Single SWF Systems

There are 2 main ways to use Flash for the interface of an Exsys system. The simplest is when the Exsys system has a relatively small number of input questions that can all be asked with a single Flash SWF file. In this case:

- All the input values are set in a single SWF file. This can be a single screen or a series of screens controlled by Action Script but still in a single SWF file
- All of the user input data is sent to the Corvid Servlet Runtime in a single call
- The Exsys Inference Engine analyses the data to reach conclusions
- The conclusions are sent back to the Flash application
- The same SWF file parses and displays the results

What makes this simple is that there is only a single Flash SWF file and Exsys does all the analysis at once. Exsys does not need to save any “session” information or “state” to use with subsequent questions in an interactive session.

When this type of system runs:

1. The Flash application uses the various controls and graphic options to allow the user to provide input
2. When the user has made their selections, they click a “Submit” button or similar action
3. Action Script is used to collect the input and format a URL calling the Corvid Servlet Runtime on a server, passing the user’s data
4. The Corvid Runtime loads the appropriate Corvid CVR file and uses the rules and logic in the system to process the input.
5. When complete, the Runtime system creates XML data that contains the results and recommendations which is sent back to the Flash application as XML data
6. The Flash application waits for the Corvid Runtime to return the XML data containing the results
7. The Flash application reads and parses the XML data to display the results to the user using whatever capabilities of Flash are needed

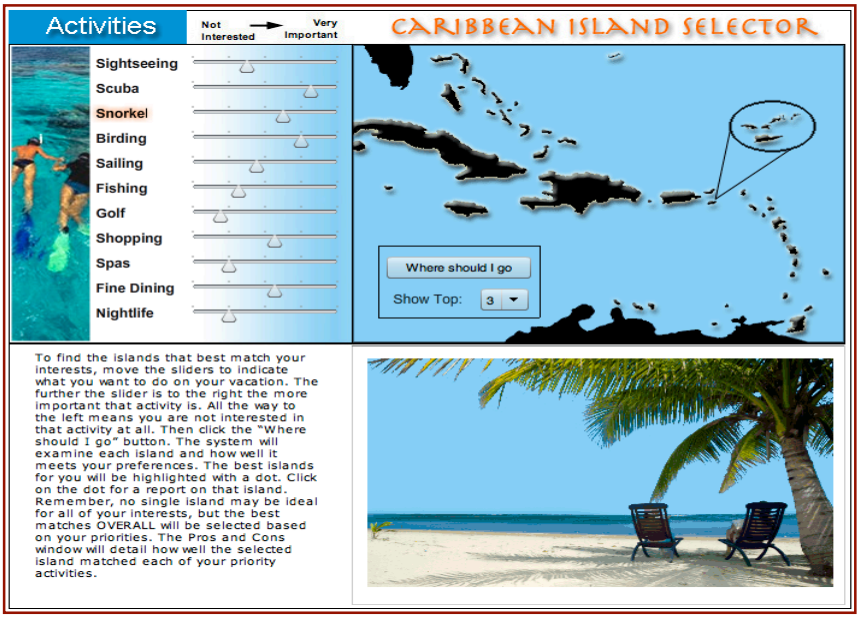
This approach can be used for many types of systems. It is practical whenever there is a fixed set of questions to ask the end user, or there is only a small amount of logic that includes/excludes certain questions. This works well for "product selection" systems where the end user sets their preferences in a standard set of input questions and the Exsys system selects the "best" products or courses of action based on weighting the user requests against various options.

For an example of this approach, look at the demo system: "Caribbean Island Selector" at:

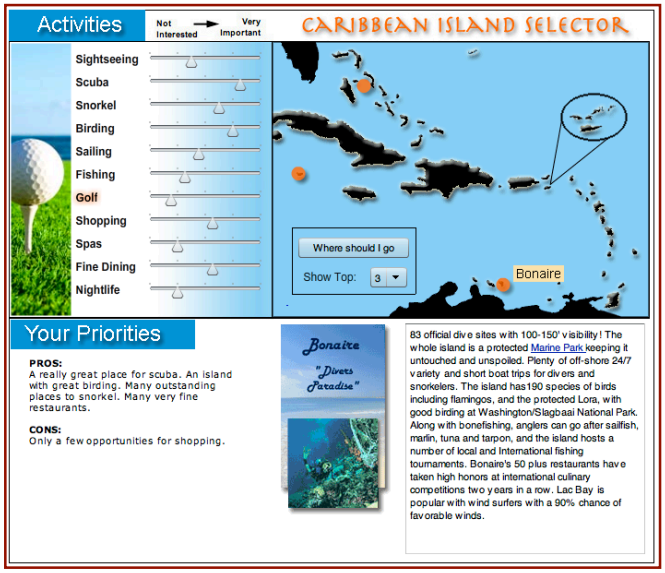
http://www.exsyssoftware.com/FlashDemos/Caribbean/IslandSelector.html

This system presents a Flash user interface with a set of sliders that the user can use to indicate their preferences for which activities are desired on the vacation.

Once the values for the various activities are set, clicking on the "Where should I go" button sends the data to the Exsys Runtime. A Corvid system using Metablocks ranks each island in a spreadsheet based on how well it matches (or does not match) the user's preferences. Exsys builds a sorted list of the best islands along with a pros/cons list for how appropriate the island would be for the user. This data is sent back to the Flash application as XML data. Action Script is used to parse the XML data and display orange circles on the "best" islands. A click on each of the circle buttons displays the details on that island at the bottom of the window.



Once the data has been run, the "Show Top" drop-down can be used to display more or less islands based on the activity values used for the run. This is done by just displaying more or fewer islands using the existing XML data rather than rerunning the system.



Multiple SWF Systems

Many systems are too complex for a single SWF file. This is particularly true when the order of the questions is highly dynamic and controlled by the logic of the rules. In these cases, it would be impractical (and redundant) to try to program the logic that controls the question order into a single SWF. There may also be many different types of questions that make use of quite different features of Flash. It would be possible to build single large SWF, but it would quickly become a development and maintenance nightmare, and probably result in an unnecessarily large SWF to load at system startup.

Fortunately this is not necessary, and with just a little Action Script, it is possible to have one SWF file load another and pass the necessary Exsys data into it, to continue the session. This allows multiple small SWF files to be used for different questions, or "template" SWF files can be used to dynamically parse the XML data from Corvid to ask various questions with a common look-and-feel.

There are 2 main uses for Flash SWF files in a system - to ask for input and to present results. In the first example above, a single SWF did both, but normally a SWF would do one or the other.

When a Flash interface is used to ask a question, the Corvid Servlet Runtime will send it XML data on the variable(s) being asked. This includes prompt text, allowed values, and range limitations. This XML data enables the SWF to modify its content to match the question and validate the input before sending it back to the Corvid Runtime. It is also possible to simply hard code all the text in the Flash SWF question screen and ignore the XML data from Corvid, but generally it is better to use the XML data as it opens up many capabilities and makes systems easier to maintain and enhance. For example, the Corvid Servlet Runtime will send the prompt text for the variable. If the SWF file uses the XML data to determine the prompt text for the Flash application, then editing the text need only be done in the Corvid development environment and it will automatically be picked up and used in the Flash interface.

If the system is to run in multiple languages, Corvid can use resource files to select the appropriate language prompt text, send it to the SWF and have a single SWF run in multiple languages. The text for values in Static Lists are also passed in the XML and can be used in the same way. When there are multiple question screens with similar layout, it is possible to build a single Flash SWF file that reconfigures the text and number of buttons for each question.

The Exsys Corvid v5 development interfaces makes it easy to associate a Flash SWF file with an Exsys variable, and to specify what XML data is passed to the SWF file. This allows the logic of a system to dynamically determine the order of the questions to ask. The XML data for each question will indicate the SWF to use.

The dynamic nature of parsing the XML data to determine screen layout makes the Action Script a little more complicated, but for many systems the system maintenance benefits are far greater.

Exsys / Flash Interaction Overview

There are 3 steps in the process of connecting Flash SWF files with the Corvid Servlet Runtime.

1. Sending Data to the Corvid Servlet Runtime

Each of the question SWF files will send data to the Corvid Servlet Runtime. This may be all the data needed to run the system, or the value for a single question. Each question screen sends data to the Runtime using POST.

The FIRST post to the Corvid Servlet Runtime MUST be a string of the form:

RUNMODE=FLASH&KBNAME=ExsysKBLocation&[var1]=value1& [var2]=value2...

All subsequent posts must be of the form:

RUNMODE=FLASH&~SCRNUM=scrnum&[varX]=valueX& [varY]=valueY...

Note: the first call has the second parameter pair "KBNAME=ExsysKBLocation" All subsequent posts have the second pair " ~SCRNUM=scrnum"

The "RUNMODE=FLASH" tells the Corvid Servlet Runtime this is from a Flash SWF file, and that the response should be in XML, rather than the normal HTML that is used in non-Flash systems.

The "KBNAME=*ExsysKBLocation*" indicates which Corvid CVR file to run. This is the same as when running non-Flash systems. This MUST be the second pair for the first post to Corvid.

The "~SCRNUM=*scrnum*" tells the Corvid Servlet Runtime which session is being run, and where the system is in the session. **This is required for all posts except the first.** The value of *scrnum* will be provided in the XML data sent back from the previous call to the Corvid Servlet Runtime. (This is explained in more detail when looking at the actual code and XML data.)

The variable / value pairs are also the same as in non-Flash systems. The variable name must be in [ ] followed by "=" followed by the value(s). For Static List variables, if there are multiple values they should be separated by commas. There can be as many variable/value pairs as needed but each pair must be separated by "&".

Once the string of data to send to Corvid is built, it is sent using POST.

Normally it is a good idea to specify the URL for the Corvid Servlet Runtime and Corvid CVR system file being run as variables so they can be easily modified if the system is moved. (If it is impractical to rebuild the SWF when files are moved, these can be put in a file that is read by Action Script commands in the SWF file.)

```
var ExsysRuntimeLocation:String = "http://www.myServer.com/CORVID/corvidsr";
var ExsysKBLocation:String = "../FlashSystems/mySystem.cvR";
var header:URLRequestHeader = new URLRequestHeader("pragma", "no-cache");
```

Create a URLRequest to the Corvid Servlet Runtime using the ExsysRuntimeLocation variable.

```
var request:URLRequest = new URLRequest(ExsysRuntimeLocation);
```

Build the data to send in a string. For example:

```
dataToSend = "RUNMODE=FLASH&KBNAME=”+ ExsysKBLocation+”&[Color]="+numSel;
```

Remember, subsequent SWF files MUST include ~SCRNUM= followed by the screen number sent back from Corvid in XML. This is not needed (or possible) for the first screen, but is required for all other screens.

Set the request.data to the data string that was built

```
request.data = dataToSend;
```

A URLLoader object is created to send the data.

```
var loader:URLLoader = new URLLoader();

loader.dataFormat = URLLoaderDataFormat.TEXT;
```

“Listeners” must be configured for the URLLoader. The only required listener is for the COMPLETE event because that is when the XML data sent back from Corvid will be available, but others can be used as desired to show progress, check for errors, etc.

```
configureListeners(loader);

function configureListeners(dispatcher:IEventDispatcher):void {
        dispatcher.addEventListener(Event.COMPLETE, loadCompleteHandler);
}
```

Set the method to POST and send the header.

```
        // call Exsys on the server
request.method = URLRequestMethod.POST;
request.requestHeaders.push(header);
```

Try to load the request, which will pass the data to the Corvid Servlet Runtime, catching and displaying any error that might occur.

```
try {
   loader.load(request);
} catch (error:Error) {
   trace("Unable to send data to Corvid Servlet Runtime: "  + error);
}
```

2. Using the XML Data Sent Back from the Corvid Servlet Runtime

The Corvid Servlet Runtime will process the data sent, using the logic and rules in the Corvid system. This may lead to the Corvid Inference Engine determining that more data is needed, or it may be able to reach a conclusion and display the results and advice. In either case, it will send back XML data.

The "loadCompleteHandler" specified in the Event Listeners will be called when the COMPLETE event for the load occurs. This processes the data sent back from Corvid:

```
function loadCompleteHandler(event:Event):void {
   var exsysData:URLLoader = URLLoader(event.target);
```

The nextScreenXMLData will hold the XML data returned from Exsys. At this point the data will be used mainly to determine the next SWF to load and action to take.

```
var nextScreenXMLData:XML;

if (exsysData.data) {      // if there is data returned
      try {
         var returnedData:String;    // data returned from Exsys

         returnedData = exsysData.data;

         nextScreenXMLData = XML(returnedData);
```

This now converts the data sent back from Corvid to an XML object, which can be used to determine what to do, which SWF file to use next, how to reconfigure controls, etc. Corvid can send a variety of information based on the situation and parameters set by the system developer. (These are explained in more detail below). There are a few items of data that are needed for most systems. The easiest way to use the XML data is via Flash's E4X syntax.

The first thing to check is for errors. If the Corvid system detected any error in processing the data, it will set "nextScreenXMLData.action.type" to "ERROR" with the explanation of the error in "nextScreenXMLData.error" to the error string. If this occurs there is some problem in the system that needs to be examined and fixed.

```
if (nextScreenXMLData.action.type == "ERROR") {          // if an error was returned
         trace(nextScreenXMLData.error);                 // display the error
         return;                                         // exit
}
```

If nextScreenXMLData.error is an empty string (""), then there was no error and the returned data can be used. The first thing to determine is the type of action to take next. This is also in "nextScreenXMLData.action.type". There are various possible values (explained below), but the most common are "ASK" and "RESULTS". The value "ASK" indicates that there is another question to ask, and "RESULTS" indicates that the system is done and the XML included the results/advice to display.

```
if (nextScreenXMLData.action.type == "ASK")
        // display the next screen

if (nextScreenXMLData.action.type == "RESULTS")
        // display the results / advice
```

If either the next question or results requires loading another SWF file, the name of that file will be in "nextScreenXMLData.action.template". In that case, a function to load the next screen is called, passing it the name of the SWF file to use and the data returned from Corvid. This can be used to set the text and layout of the next screen, along with setting the value for "~SCRNUM" that may need to be passed later.

```
loadNextScreen(nextScreenXMLData.action.template, returnedData);
```

3. Communication Between SWF Files

When a system uses multiple SWF files, the XML data sent back from Corvid specifies the SWF to load next. That SWF must be loaded and the data passed to it. This is probably the most complicated aspect of integrating Flash with Corvid because it uses capabilities of Flash not typically used for most Flash applications, and it requires using events in a precise (somewhat non-intuitive) way or the technique will not work. The data is sent between SWF files with the Action Script 3 LocalConnection object. (As with many things in Flash, there may be other ways to achieve the same result - but this is the approach we recommend)

Each SWF should have a sending and receiving LocalConnection object

```
var sendingLC:LocalConnection;
var receivingLC:LocalConnection;
```

The sendingLC is used to send data to the next SWF, and the receivingLC is used to get data from a previous SWF. Since the starting SWF does not receive data, it only really needs the sendingLC, and since the last SWF (typically for results display) does not send data to continue the session, it only really needs the receivingLC.

In Section #2, when the XML data returned from Corvid was read, loading the next SWF file was down using loadNextScreen(). This is the function that loads the SWF and passes the Corvid data to it.

```
function loadNextScreen (fileToLoad:String, dataToPass:String) {
```

This is done by creating a new Loader object.

```
var nextScreenLoader:Loader = new Loader();
```

This Loader requires two event listeners - one for INIT and one for COMPLETE. The problem here is that as the new SWF is loaded, the old one becomes inactive. Data has to be sent with LocalConnection in the brief interval when the old SWF is still active and can send data, and the new SWF has become active enough to receive the data. The trick to doing this is to establish the connection on the Loader's INIT event, but to send the data on the COMPLETE event.

On the next SWFs INIT event establish the LocalConnection from this SWF for “CorvidData”. This MUST be done on the INIT event for the data to be successfully passed. If you wait until the COMPLETE event the connection cannot be setup because the previous SWF is gone.

The INIT part is:

```
nextScreenLoader.contentLoaderInfo.addEventListener (Event.INIT ,
    function(evt:Event):void {
            sendingLC = new LocalConnection();
            sendingLC.client = this;
             try {
                sendingLC.connect("CorvidData");
            } catch (e:Error) {
                trace("Local Connection error : " + e);
             }
     }

);
```

(For those that prefer to create a separate named function, that will also work. It is included internally here to make it clearer that it is associated with the INIT event.)

This INIT function will fire as soon as the next SWF is loaded and initialized, but not displayed.

On the COMPLETE event for loading the next SWF, call addChild to display that SWF and send the data to that screen. The INIT event will have already established the LocalConnection. The new SWF is responsible for picking up the passed data and processing it.

```
nextScreenLoader.contentLoaderInfo.addEventListener (Event.COMPLETE,
    function(evt:Event):void {
            addChild(nextScreenLoader);
```

Send the data return from Corvid (dataToPass) labeled “CorvidData” to be processed by the function “NextScrn” when it is read by the next SWF file.

```
             sendingLC.send("CorvidData", "NextScrn", dataToPass)

    }
);
```

Once the event Listeners are added, go ahead and load the next screen.

```
            // load the screen
nextScreenLoader.load(new URLRequest(fileToLoad));

}
```

This process of loading new SWF files and passing data can be repeated as many times as needed.

For the final screen, typically when action.type = “RESULTS”, the SWF to display the results screen should be loaded. This will not need a sendingLC since it is the last screen. The result data will be passed in the XML, and the screen will need to parse and display it.

For all SWF except the first, when the SWF loads it should create the recievingLC and look for data.

```
receivingLC = new LocalConnection();
receivingLC.client = this;
```

Try to receive the data sent by the previous SWF file under the label “CorvidData”. If this fails, report an error.

```
try {
        receivingLC.connect("CorvidData");

} catch (error:Error) {
        trace( "Receiving LocalConnection Error: Can't connect");
}
```

Once the “CorvidData” is read, the function specified in the LocalConnection.send from in the previous SWF file will be called. In the example, this is the function “NextScrn”. (The names “CorvidData” and “NextScrn” can be anything you choose, as long as they are used consistently)

The NextScrn function will automatically be passed the data sent as “CorvidData”. It converts into the global Flash XML object, receivedXMLData, for use later and closes the local connection.

```
function NextScrn(passedData:String):void {   // name specified in LocalConnection call

    receivedXMLData = new XML(passedData);
    receivingLC.close();

    buildUI();   // build the UI based on the passed data

}
```

The function buildUI() uses the XML data to layout the controls, set text or do whatever is needed to present the user interface based on the data from Corvid. For customized SWF files designed for specific questions, this may not require any action. For “template” SWF files, it may require creating and laying out controls based on the type of question(s) and number of possible values.

Once the user provides their input, the data is sent back to corvid using the same approach as in section #1 above. However, for each SWF except the first, the data POSTed to Corvid MUST include the ~SCRNUM parameter. The value for this comes from receivedXMLData. The string value that must be passed as ~SCRNUM is

receivedXMLData.scrnum

The post string will be similar to:

“RUNMODE=FLASH&**~SCRNUM=” + receivedXMLData.scrnum** + “&[var1]=” +...

The Action Script Code in Detail

This Action Script code can be copied into a system. A copy of it is installed with Corvid in the folder:

/ProgramFiles/Exsys/Corvid/SampleFlashActionScript.txt

Required imports to handle Flash events, and call URLs

```
import flash.events.*;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestHeader;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
```

Required import of the LocalConnection object to pass data between SWF files.
(If there is only a single SWF file for the entire system, this LocalConnection import and all the objects based on LocalConnection are not needed.)

```
import flash.net.LocalConnection;          // Send and receive data
```

Add imports for whatever controls will be used in the system

```
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;
```

Define strings for the location of the Exsys Servlet Runtime and Exsys application CVR file. These can just be coded into the system at the appropriate place, but putting them here makes it easy to find when the URL changes or the system is moved to a new server. If you do not want to recompile the Flash code whenever the system is moved, these locations can be put in a text file that would be read at the start of the run. In that case, only the text in the file would need to be changed when systems are moved, without rebuilding the SWF file.

```
var ExsysRuntimeLocation:String = "http://www.myServer.com/CORVID/corvidsr";
var ExsysKBLocation:String = "../FlashSystems/mySystem.cvR";
```

Define LocalConnection objects to send and receive data. The first screen only needs a sendingLC. The last screen needs only a receivingLC to get data from the previous screen. All other screens need both. If a system has only a single SWF file, these LocalConnection objects are not needed.

```
var sendingLC:LocalConnection;   // First screen only needs to send.  Others need to send and receive
var receivingLC:LocalConnection;  // Last screen only needs to receive.  Others need to send and receive
```

Define a global variable receivedXMLData to hold the XML data returned from Exsys and passed to the current SWF from the previous SWF file. This is not needed in the first SWF file, or for systems that only use a single SWF file.

```
var receivedXMLData:XML;      // The XML data passed in
```

Setup the receiving LocalConnection when this Action Script starts. This is not needed in the first SWF file, but must be included in all others when there are multiple SWF files.

```
receivingLC = new LocalConnection();
receivingLC.client = this;
```

Try to receive the data sent by the previous SWF file under the label “CorvidData”. If this fails, report an error. This is not needed in first SWF or if there is only a single SWF.

```
try {
        receivingLC.connect("CorvidData");

} catch (error:Error) {
        trace( "Receiving LocalConnection Error: Can't connect");

}
```

This is the function specified by LocalConnection.send from in the previous SWF file. This function will be passed the data sent. It just converts it into the global Flash XML object, receivedXMLData, for use later, closes the local connection and calls the buildUI() function to use the XML data to layout the controls. This is not needed in first SWF or if there is only a single SWF.

```
function NextScrn(passedData:String):void {   // name specified in LocalConnection call
        receivedXMLData = new XML(passedData);
        receivingLC.close();
        buildUI();   // build the UI based on the passed data

}
```

The buildUI() function (or whatever name is preferred) uses the passed XML data to configure the user interface - set Prompt text strings, add labels to buttons, arrange components, etc. The degree to which this is needed depends on how specific/generic the SWF file is. Very general purpose SWF files may do significant reconfiguration to work with various questions. SWF files dedicated to a specific question may not need to do anything.

```
function buildUI():void {
// create and arrange components
}
```

Add handler(s) to send data to Exsys. This sample system assumes a simple design with 2 radio buttons and an “OK” button. The system will send the value of the selected radio button to Exsys. A real system can have as many controls and control types as desired. The point here is that the OK button (or some other action) indicates that the input is complete and data is to be sent to Exsys.

This event handler detects a click on the OK button (labeled “btnOK”) to build the data string to send to Exsys. This may involve collecting data from multiple controls/objects.

```
btnOK.addEventListener(                          // if the OK button is clicked
        MouseEvent.CLICK,
        function processInput() {

                var numSel:uint;                 // number of the radio button selected
                var dataToSend:String = "";      // string that will be sent to Exsys

                if (radio1.selected == true)     // if radio button 1 set value to 1
                  numSel = 1;
                else                             // otherwise set the value to 2
                  numSel = 2;
```

Once data is collected, it will be sent to the Corvid runtime for processing. This is done with a URLLoader.

```
                var loader:URLLoader = new URLLoader();

                //setup loader for next screen

                loader.dataFormat = URLLoaderDataFormat.TEXT;
```

Configure the listeners for the URLLoader. (Done below) The only required listener is for the COMPLETE event, but others can be used as desired to show progress, check for errors, etc.

```
                configureListeners(loader);
```

Set the header to not cache data. (This may not be required in all cases, but could prevent some problems.)

```
                var header:URLRequestHeader = new URLRequestHeader("pragma", "no-cache");
```

Point a URLRequest to the Corvid Runtime using the ExsysRuntimeLocation variable.

```
                var request:URLRequest = new URLRequest(ExsysRuntimeLocation);
```

The data to send to Corvid should be setup as a series of name=value pairs separated by &.

The first parameter passed to Corvid MUST be RUNMODE=FLASH. This tells Exsys to return XML data rather than the normal HTML.

For the first call to Exsys, the second pair MUST be the KBNAME=name. This can be done using the ExsysKBLocation variable defined at the top of the code. For subsequent calls to Exsys, the second pair MUST be:

~SCRNUM = receivedXMLData.scrnum

where the receivedXMLData will be sent back from Exsys. This passes back the screen number used by Exsys.

After that as much data as desired can be added using [varname]=value separated by &. This can be repeated for multiple variables. The variable name MUST be in [ ]. The variable name can be obtained from the XML data that Exsys sends or hard coded. For the first SWF file it must be hard coded since no data is normally passed in.

In this example, the Exsys variable [Color] is assigned the value selected from the radio button.

```
dataToSend = "RUNMODE=FLASH&KBNAME=”+ ExsysKBLocation+”&[Color]="+numSel;
```

Set the request.data to the value string that was built

```
request.data = dataToSend;
```

Set the method to POST and send the header.

```
        // call Exsys on the server
request.method = URLRequestMethod.POST;
request.requestHeaders.push(header);
```

Try to load the request, which will pass the data to Corvid, catching and displaying any error that occurs.

```
try {
   loader.load(request);
} catch (error:Error) {
   trace("Unable to post data to Corvid Servlet Runtime."  + error);
}
```

Once loader.load() is called, the system has to wait for Exsys to process the data and send back the results. This may be only moments or some time, depending on server, system size, number of rules, etc. The listeners assigned to the loader will activate when it is done (or warn if it fails). In the meantime, clean up the screen and if desired, display some form of “in process” graphic to let the user know the system is running.

While waiting for Exsys to return data, remove or hide any controls used on the screen that are unique to the specific question being asked. If the control is to be reused in a template, it can be hidden or otherwise obscured. The control that sends the data to Corvid should be immediately removed or disabled to prevent sending multiple sets of data.

```
                    // remove or hide controls from this screen
                      myPrompt.visible = false;
                      removeChild(radio1);
                      removeChild(radio2);
                      removeChild(btnOK);
}
);
```

This configures the listeners for the loader object. Only the COMPLETE event is required, but the others can be used to report problems or indicate progress to the user.

```
// only the COMPLETE event is required.  Others can be used as desired
function configureListeners(dispatcher:IEventDispatcher):void {
    dispatcher.addEventListener(Event.COMPLETE, loadCompleteHandler);
    dispatcher.addEventListener(Event.OPEN, openHandler);
    dispatcher.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    dispatcher.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
    dispatcher.addEventListener(HTTPStatusEvent.HTTP_STATUS, httpStatusHandler);
    dispatcher.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
}
```

The loader COMPLETE event gets the XML data sent back from Corvid, stores it and creates and XML object from it that can be used in other parts of the system.

```
function loadCompleteHandler(event:Event):void {
    var exsysData:URLLoader = URLLoader(event.target);
```

The nextScreenXMLData will hold the XML data returned from Corvid. At this point the data will be used mainly to determine the next SWF to load and action to take.

```
    var nextScreenXMLData:XML;


    if (exsysData.data) {      // if there is data returned
            try {

                var returnedData:String;    // data returned from Exsys

                returnedData = exsysData.data;

                nextScreenXMLData = XML(returnedData);


                    // parse XML data to decide what to do next
```

The XML data returned from Exsys has an Action Type value that can be used to determine what to do next. This should be parsed and used to determine what to do. (More on the XML below)

If the XML Action Type is “ERROR”, check the “error” value. It should be displayed and processing stopped. This will be an Exsys error message.

```
                if (nextScreenXMLData.action.type == "ERROR") {          // if an error was returned
                    trace(nextScreenXMLData.error);   // display the error
                    return;                                    // exit
                }
```

If the action.type from Corvid is ASK, then load the next screen, or reconfigure the controls on a template screen. Send the returnedData string on to the next screen, which will build its own XML object and use it as needed.

When there is an ASK command, the action.template value will be the SWF file to use. This can be specified for the variable in the Corvid development environment.

```
if (nextScreenXMLData.action.type == "ASK")
    loadNextScreen(nextScreenXMLData.action.template, returnedData);
                    // load screen and send data
```

The action type can also be “RESULTS” in which case the XML data will contain the system recommendations and advice. An Action Type of “Display_HTML” indicates a URL should be immediately displayed.

```
        } catch (e:Error) {
            trace("XML error : " + e);
        }


    }

}
```

Other optional loader events. The errors should be sent to trace() which will display them.

```
function openHandler(event:Event):void {
}

function progressHandler(event:ProgressEvent):void {
  trace("progressHandler loaded:" + event.bytesLoaded + " total: " + event.bytesTotal);
}

function securityErrorHandler(event:SecurityErrorEvent):void {
  trace("securityErrorHandler: " + event);
}

function httpStatusHandler(event:HTTPStatusEvent):void {
  trace("httpStatusHandler: " + event);
}

function ioErrorHandler(event:IOErrorEvent):void {
  trace("ioErrorHandler: " + event);
}
```

Load the next SWF file. If the template approach is used, this should instead reconfigure the controls in the existing SWF file.

```
            // load the next screen and pass the XML data
        function loadNextScreen (fileToLoad:String, dataToPass:String) {

            var nextScreenLoader:Loader = new Loader();
```

Add an INIT event listener for the SWF file that is being loaded. This will fire as soon as the next SWF is loaded and initialized, but not displayed. It is VERY IMPORTANT that this be done on the INIT event. There is only a moment when both SWF files exist in a way that allows data to be sent and received.

On INIT the new SWF must setup the LocalConnection.

```
                    // on INIT make Local Connection
    nextScreenLoader.contentLoaderInfo.addEventListener (Event.INIT ,
        function(evt:Event):void {
            sendingLC = new LocalConnection();

            sendingLC.client = this;
```

On the next SWFs INIT event establish the LocalConnection from this SWF for “CorvidData”. This MUST be done on the INIT event for the data to be successfully passed. If you wait until the COMPLETE event the data will be gone.

```
            try {
                sendingLC.connect("CorvidData");
            } catch (e:Error) {
                trace("Local Connection error : " + e);
            }
        }

    );
```

On the COMPLETE event for loading the next SWF, call addChild to display that SWF and send the data to that screen. The INIT event will have already established the LocalConnection. The new SWF is responsible for picking up the passed data and processing it.

```
                    // on COMPLETE add the SWF file and send the data
    nextScreenLoader.contentLoaderInfo.addEventListener (Event.COMPLETE,
        function(evt:Event):void {
            addChild(nextScreenLoader);
```

Send the data labeled “CorvidData” to be processed by the function “NextScrn” when it is read by the next SWF file. (These names can be anything you choose, as long as they are used consistently)

```
            sendingLC.send("CorvidData", "NextScrn", dataToPass)

        }
    );
```

Once the event Listeners are added, go ahead and load the next screen.

```
                    // load the screen
    nextScreenLoader.load(new URLRequest(fileToLoad));

    }
```

The new SWF screen will be loaded and the process can be repeated as needed. When the action.type = “RESULTS”, then the SWF to display the results screen should be loaded. It will parse the normally more complex result XML data and display it.

The XML Data Sent by the Corvid Servlet Runtime

Whenever a Flash SWF posts data to the Corvid Servlet Runtime, Corvid sends back XML data that can be used by the SWF. This XML data can be easily converted to a Flash XML object with:

```
nextScreenXMLData = XML(returnedData);
```

and then incorporated into your Flash code using Flash’s E4X syntax.

E4X provides a very convenient way to refer to the XML data that Exsys sends and even allows you to extend it with your own XML tags unique to your system.

The type of data varies with the action being performed, and the options selected in the development environment.

The data Exsys sends is always wrapped in an **<exsysdata>** tag. There are some additional tags that are always sent, along with some optional ones.

The XML data will always be:

<exsysdata>
<scrnum> *Exsys Screen Identifier* **</scrnum>**
<action>
<type> *Action to take ASK, RESULTS, DISPLAY_HTML or ERROR* **</type>**
<target> *Target for the Action* **</target>**
<template> *SWF file to use for the next action* **</template>**
</action>

<error> *Any Error Text or blank if there is no error* **</error>**

<exsysVar>
Data on one or more Exsys Variables
</exsysVar>

<trace> *trace data if that is turned on* **</trace>**

</exsysdata>

<scrnum>

Inside the <exsysdata> tag, there will always be a <scrnum> tag:

<scrnum> Exsys Screen Identifier </scrnum>

This is added automatically by Exsys. Exsys uses the <scrnum> value to tell it what session it is in and where it is in that session. All POSTs back to Exsys (except the first) MUST include:

~SCRNUM=*value of <scrnum>*

as the second item the POST. Exsys will automatically use this value to find the appropriate session and update the <scrnum> value when it sends XML data back for the next SWF file. Each set of data returned from Corvid will have a different scrnum value, so it must be obtained each time. The value of scrnum can be obtained from the nextScreenXMLData object using E4X as:

receivedXMLData.scrnum

<action>

Inside the <exsysdata> tag, there will always be a <action> tag. This is to indicate what type of action is to be performed next and how to perform it. The <action> tag always has 3 parts:

<action>
<type> *Action to take ASK, RESULTS, DISPLAY_HTML or ERROR* **</type>**
<target> *Target for the Action* **</target>**
<template> *SWF file to use for the next action* **</template>**
</action>

The <type> tag indicates the type of action is to be performed using the other tags. The possible values are:

ASK	Ask the user the the value of the Corvid variable named in <target> using the SWF file specified in <template>
RESULTS	Display the results using the SWF specified in <template>
DISPLAY_HTML	Display the html page specified in <target>
ERROR	An error occurred. Display the error message in receivedXMLData.error

It is the responsibility of the developer to check the action type to determine what to do. Even in cases where it is known what should follow the current question, there should always be a check for <type> being "ERROR":

if (receivedXMLData.action.type == "ERROR") {
report and handle the error in receivedXMLData.error
}

<error>

Inside the <exsysdata> tag, there will always be a <error> tag. This will normally be blank, but if an error occurred in processing the data sent to the Corvid Servlet Runtime, it will be reported here. If the error tag is not blank, the action type tag will be "ERROR". All returned data should be checked to see if the error tag is not blank, or the action type tag is "ERROR". It there is an error, it should be reported or displayed in some way so it can be corrected and that session terminated.

<trace>

A <trace> tag will be added to the <exsysdata> ONLY if the system has "Add Trace in Servlet" selected in the Corvid development environment. This should be used ONLY for debugging during development since it will add a large amount of text to the XML file and make the system run MUCH slower than normal. However, if a system is not producing the correct results, this is an easy way to get the trace for the run. If the contents of receivedXMLData.trace are put in an edit box, it can be easily copied and pasted into an editor to examine it.

The <exsysvar> tag

The <exsysvar> tag is used to provide information on a Corvid variable. This may be to provide the information needed to ask the value of the variable in an ASK action, or to provide the final value set as part of the advice and recommendations in a RESULTS action. There can be multiple <exsysvar> tags in the XML data when that is needed to ask multiple questions on the same screen, or when the results require the value of multiple variables.

The <exsysvar> can have the following tags in it. (Not all tags are used in all cases.)

<exsysVar>
<name> *Variable name* **</name>**
<type> *Variable type* **</type>**
<varPrompt> *Prompt* **</varPrompt>**

<possibleValues>
<count> *Number of possible values for List variables* **</count>**
<valueText> *text of value 1* **</valueText>**
<valueText> *text of value 2* **</valueText>**
<valueText> *text of value 3* **</valueText>**
...
</possibleValues>

<defaultValue> *Default value* **</defaultValue>**

<control> *Type of Control to use when asking* **</control>**

<alsoAsk>
<count> *Number of also ask variables* **</count>**
<alsoAskVarName> *Name of "Also Ask" variable 1* **</alsoAskVarName>**
<alsoAskVarName> *Name of "Also Ask" variable 2* **</alsoAskVarName>**
<alsoAskVarName> *Name of "Also Ask" variable 3* **</alsoAskVarName>**
...
</alsoAsk>

<limits>
If "Numeric":
<upper> *Upper limit value or blank* **</upper>**
<lower> *Lower limit value or blank* **</lower>**
<intOnly> *Require integer value -TRUE or FALSE* **</intOnly>**

If "StaticList" or "DynamicList":

<maxNumValues> *Number of allowed values* **</maxNumValues>**

If "String":

<mask> mask string or blank </mask>

If "Date":

<maxDaysPast> *Max days past value or blank* **</maxDaysPast>**

<maxDaysFuture> *Max days future value or blank* **</maxDaysFuture>**

</limits>

<currentValue> *The current value set for the variable* **</currentValue>**

<other> *Other info the developer sends - could be other XML tags* **</other>**

</exsysVar>

<name>

Within an <exsysVar> item, there will always be a <name> tag with the name of the Corvid variable. This will NOT be in [ ].

<name> ***varname*** **</name>**

In cases where there are multiple <exsysvar> tags in the XML to describe multiple variables, the name tag is the most convenient way to refer to individual variables with E4X. For example, to get the prompt for the Corvid variable [color] :

nextScreenXMLData.exsysVar.(name == "*color*").varPrompt

To get the current value of the Corvid variable [color] :

nextScreenXMLData.exsysVar.(name == "*color*").currentValue

When the nature of the system guarantees that there will be only a single <exsysvar> tag, this is not needed and a simpler format can be used:

nextScreenXMLData.exsysVar.varPrompt

<type>

Within an <exsysVar> item, there will always be a <type> tag with type of variable. The type is a string and can be:

StaticList
DynamicList
Numeric
String
Date
Collection
Confidence

Note that “StaticList” and “DynamicList” do NOT have spaces in them. The type can be used to determine how to ask a question, the options on values and the tags that may be in the <limits> tag.

<varPrompt>

Within an <exsysVar> item, there will always be a <varPrompt> tag with the prompt to use when asking for the variable. If a Corvid variable has multiple prompts or resource files are being used, the correct prompt will have already been set by Corvid.

<possibleValues>

The <possibleValues> tag provides the information on the values that can be selected among for questions that ask Static or Dynamic List variables. This tag is included only if the <type> tag is either “StaticList” or “DynamicList” and the action type is “ASK”.

Within the <possibleValues> tag there is a <count> tag with the number of values that can be selected among. There are then a series of <valueText> tags each with the text of one of the possible values. Template SWF files can check the <count> tag and then read and display that many <valueText> values.

As with prompts, multiple value or resource file settings will have already been resolved by Corvid before creating the XML. For Dynamic Lists, the value list will have been populated from the variable’s associated source.

<possibleValues>
<count> ***number of values*** **</count>**
<valueText> ***value 1*** **</valueText>**
<valueText> ***value 2*** **</valueText>**
<valueText> ***value 3*** **</valueText>**
...
</possibleValues>

<defaultValue>

Within an <exsysVar> item, there will be a <defaultValue> tag when the action type is “ASK”. This is the default value to use for the variable when asking the user for a value. The option to use a default value, and the value itself, is set in the Corvid development environment.

For Static and Dynamic List variables, this will be the **number** of the value to use as the default. For String, Numeric and Date variables, this will be the text of the value to use as the default. If there is no default value specified for the variable, this will be blank.

<control>

Within an <exsysVar> item, there will be a <control> tag when the action type is “ASK”. This will contain the “Control to Use” that was specified for the variable in Corvid. This can be used for more complex template SWF files that can accommodate multiple types of controls. (A typical use would be to select between checkboxes and radio buttons.) Flash SWF files that are designed for a single control type can ignore the value of this tag.

The values for <control> are a number indicating the control specified in Corvid:

1	Radio Button
2	Checkbox
3	List Box
4	Drop Down List
5	Edit Box
6	Button
7	Image Map

<alsoAsk>

Within an <exsysVar> item, there will be a <alsoAsk> tag when the action type is “ASK”.

This lists any variables specified in the main variable’s Also Ask list. The main variable is the one specified by the Action Target tag. The <count> tag will have the number of variables in the list, followed by individual <alsoAskVarName> tags with the names of the variables in the list. If there are no Also Ask variables for the main variable, all will be blank.

<alsoAsk>
<count> ***number of also ask variables*** **</count>**
<alsoAskVarName> ***var 1*** **</alsoAskVarName>**
<alsoAskVarName> ***var 2*** **</alsoAskVarName>**
<alsoAskVarName> ***var 3*** **</alsoAskVarName>**
...
</alsoAsk>

If there is an Also Ask list, there will be separate <exsysVar> tags for each of the variables in the list, which can be used to get the parameters for each variable. The data for each variable can be obtained by reading the name in the list and then using E4X syntax to specify the properties for that named variable, such as:

nextScreenXMLData.exsysVar.(name == currentAlsoAskVar).varPrompt

<limits>

Within an <exsysVar> item, there will be a <limits> tag when the action type is “ASK”. This provides information on any limits to the input values that should be accepted. These limits come from Corvid and are set in the Corvid development environment. The Corvid Servlet Runtime will reject any value not meeting the limits, but this requires sending data to Corvid only to have the question reasked. It is generally better and more efficient to validate the value in Flash with Action Script before sending it to Corvid. This makes for a better user experience, and allows the SWF file to immediately display message about invalid input, or to only activate a “Submit” button when input is within acceptable ranges.

The tags within the <limits> tag depend on the variable type. If there are no limits the values will be blank.

If the variable type is ”Numeric” there will be:

<limits>
<upper> Upper limit value or blank </upper>
<lower> Lower limit value or blank </lower>
<intOnly> Accept only integers - “TRUE” or “FALSE” </intOnly>
</limits>

If there is a <upper> value, the input value must be less than or equal to that value. If the <upper> value is blank, there is no upper limit.

If there is a <lower> value, the input value must be greater than or equal to that value. If the <lower> value is blank, there is no lower limit.

If the <intOnly> value is “TRUE”, then only a value that is an exact integer will be accepted. If the <intOnly> value is “FALSE” there is not restriction on the value.

If the variable type is ”StaticList” or “DynamicList” there will be:

<limits>
<maxNumValues>value or blank </maxNumValues>
</limits>

If there is a <maxNumValues> value, that is the maximum number of values that can be simultaneously selected. If the value is blank, any number of values can be selected.

If the variable type is ”String” there will be:

<limits>
<mask> mask string or blank </mask>
</limits>

If there is a <mask> value, the string must match the pattern set by the mask. The mask pattern syntax is the same as when running without Flash and is explained in the Corvid manual. If the value is blank, any string is accepted.

If the variable type is ”Date” there will be:

<limits>
<maxDaysPast> Max days past value or blank </maxDaysPast>
<maxDaysFuture> Max days future value or blank </maxDaysFuture>
</limits>

If there is a <maxDaysPast> value, the input date input must not be more than that number of days in the past from the current date when the system is run. If the <maxDaysPast> value is blank, there is no such restriction.

If there is a <maxDaysFuture> value, the input date input must not be more than that number of days in the future from the current date when the system is run. If the <maxDaysFuture> value is blank, there is no such restriction.

<currentValue>

Within an <exsysVar> item, there will be a <currentValue> tag. This contains the current value of the variable. This is normally only used when action type is “RESULTS”, but can also be used for displaying intermediate results or during debugging. The value used is the immediate value, and does not lead to additional rules being fired to derive a value, or external sources for the value being used. If a final value is needed, use other commands in the system to derive the final value before using the <currentValue> tag. If the variable does not have a value set yet, this will be blank

This is the same value as [[**varname*.VALUE]].

<other>

Within an <exsysVar> item, there will be an <other> tag. The value of this tag can be set by the system developer in the Corvid development environment (details to follow). This tag can contain any string. It can include double square bracket replacements of other values and even other data in your own XML tags. (Remember if you use XML syntax to add your own tags, be sure that tags are nested correctly and closed correctly - otherwise you will get very difficult to debug error messages when the Exsys data is converted to a Flash XML object.)

Asking Questions

When the action is to ask a question, the XML data will be:

<exsysdata>
<action>
<type> *ASK* **</type>**
<target> *Name of the main Corvid Variable to ask* **</target>**
<template> *SWF file to use to ask the question* **</template>**
</action>

<error> *Any Error Text or blank if there is no error* **</error>**

<exsysVar>
Data on one or more Exsys Variables
</exsysVar

<trace> *trace data if that is turned on* **</trace>**

</exsysdata>

When the next action is to ask a questions, the action type tag will be "ASK".

```
if (receivedXMLData.action.type == "ASK") {
    // ask the next question
}
```

In this case, the action target tag will be the name of the main variable to ask:

varname = receivedXMLData.action.target

The SWF file to use to ask the variable will be in the action template tag:

SWFtoUse = receivedXMLData.action.template

The SWF file to use for asking a question can be specified in the Corvid development environment. If no template was specified, the default SWF will be ask_*varname*.swf. So, for the Corvid variable [color], the default SWF would be ask_color.swf and this would be the value of the action template tag unless another file had been specified.

Inside the <exsysdata> tag, there will be one or more <exsysVar> tags. These provide the information for asking questions and displaying results. There will always be an <exsysVar> tag for the main variable specified as the action target. If that variable has associated "Also Ask" variables, each of those will also have its own <exsysVar> tag with the data on that variable. The developer may also have specified that other <exsysVar> tags should be added.

The prompt value for a the main target variable can be retrieved with E4X by:

nextScreenXMLData.exsysVar.(name == nextScreenXMLData.action.target).varPrompt

Displaying Results

When displaying results, Exsys sends similar XML data as in ASK, but the list of variables is specified by the developer in Corvid. Since this is not a screen to ask questions, many of the parameters that are only needed for questions are not included.

<exsysdata>
  <action>
    <type> *RESULTS* **</type>**
    <target> </target>
    <template> *SWF file to use to display the results* **</template>**
  </action>

  <error> *Any Error Text or blank if there is no error* **</error>**

  <exsysVar>
    Data on one or more Exsys Variables to use in the results
  </exsysVar

  <trace> *trace data if that is turned on* **</trace>**

</exsysdata>

The list of variables to include in the results XML and the SWF file to use is specified in the Corvid development environment. It can be a single variable up to all the variables in the system. Each variable will have its own <exsysVar> tag. If no list of variables is specified in the RESULTS command (using "FlashXML="), all the variables in the system will be sent and the SWF file to be used will be *kbname*_results.swf

For each of the variables included, there will be an <exsysVar> tag but the tags <possibleValues>, <control>, <alsoAsk> and <limits> are NOT included since these are only for asking questions.

The <name> <type> <varPrompt> <curentValue> and <other> are always sent.

At the developer's option, other tags and their associated data can be sent. Default is to NOT send these tags. These are set in the Corvid development environment.

<full>	Full value of prompt and all values set. Same as [*varname*.FULL]
<short>	Short text of values set. Same as [*varname*.SHORT]
<time>	Time the variable was set. Same as [varname.TIME]
<age>	Milliseconds since the value was set. Same as [varname.AGE]
<how>	How the variable was set. Same as [varname.HOW] - this can be quite long
<age>	Custom string the developer may add. Can include [[ ]] replacement

Displaying an HTML Page

Some systems display their results by dynamically building an HTML page. This is stored on the server and the URL for the page is sent to the SWF file for display in a new browser window. This has the advantage that the results page can be easily printed and bookmarked with the Browser program.

In this case the XML data is very simple since it only needs to include the URL to open.

<exsysdata>
<action>
<type> DISPLAY_HTML **</type>**
<target> *The URL of the page to display* **</target>**
<template> </template>
</action>

<error> *Any Error Text or blank if there is no error* **</error>**

<trace> *trace data if that is turned on* **</trace>**

</exsysdata>

Error Reporting

If the Corvid Servlet Runtime encounters an error when processing the data that has been sent to it, the XML data it sends back will report the error.

In this case the XML data is very simple since it only needs to report the error.

<exsysdata>
<action>
<type> ERROR **</type>**
<target> </target>
<template> </template>
</action>

<error> *The explanation of the error* **</error>**

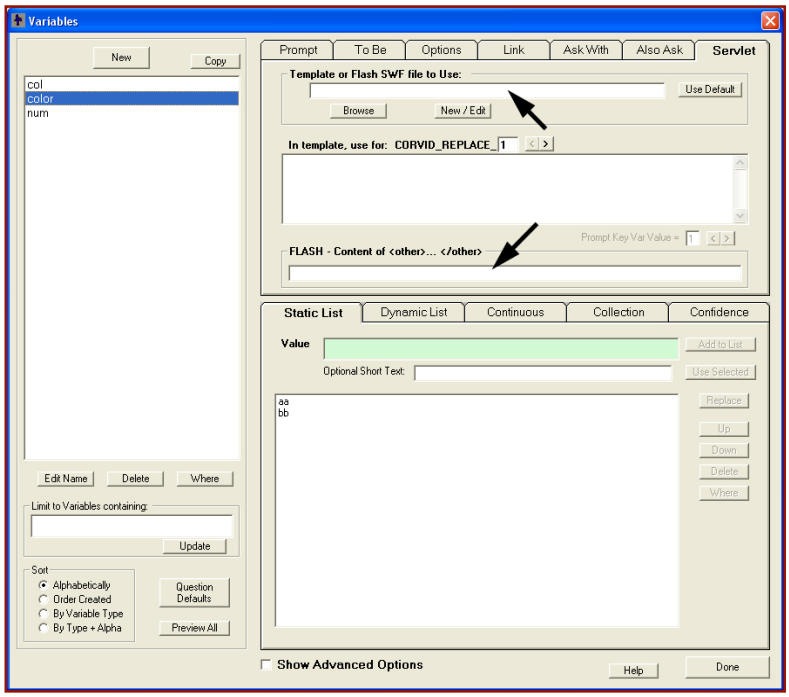
<trace> *trace data if that is turned on* **</trace>**

</exsysdata>

All XML data returned from Corvid should be immediately checked to see if the action type is “ERROR”. If it is, it should be reported in some way and the session terminated.

Specifying the Flash Interface Parameters in Corvid

In Corvid v5, several small but important changes have been made to implement the interface to the Flash. In the Variables window, there are 2 changes under the Servlet tab:



The edit box that previously was for entering the HTML template to use when running with the Corvid Servlet Runtime, can now also be used to specify the Flash SWF file to use when running with a Flash interface. When Corvid is run with Flash, the SWF file specified will be used in the XML data as the action template tag for this variable. For systems using the non-Flash HTML interface with the Corvid Servlet Runtime, the name of the HTML template to use can still be entered.

If no SWF file is entered, the file **ask_varname.swf** will be used, where “varname” is the name of the variable.

When using Flash for the user interface, it can be convenient to have the “New/Edit” button next to the edit box open the Flash or Flex IDE. To do this set the “HTML Editor” in the Properties window to point to the Flash or Flex development program. When the “New/Edit” button is clicked, the specified program will be called.

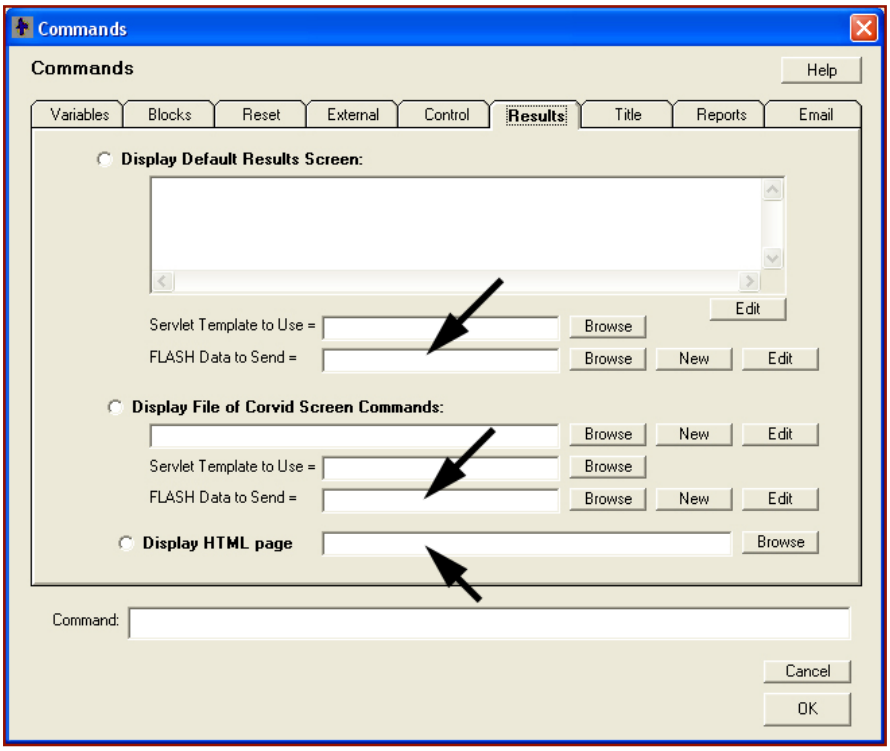
The other change is a new edit box at the bottom of the Servlet tab, which has the text to be included in the <other> tag for this variable. This allows any special text or information required by the Flash interface to be included. This will be in the <exsysVar> tag for the variable in its <other> tag. The text can be any string, and can include double square bracket embedding of other Corvid variables. The string can also use other XML tags of your design to include multiple items of data in a way that makes them easy to recover in the Flash using E4X syntax. If other XML tags are used, make SURE that all tags have matching closing tags and are correctly nested. If they are not, the Action Script commands to create XML objects from the data will fail. For example, if the “Other” text is:

<item1>aaa</item1><item2><x>123</x><y>456</y></item2>

You could recover the value of item2 x for the variable [MyVar] using:

nextScreenXMLData.exsysVar.(name == “MyVar”).other.item2.x

The Command Builder window “Results” tab has 3 changes to support Flash.



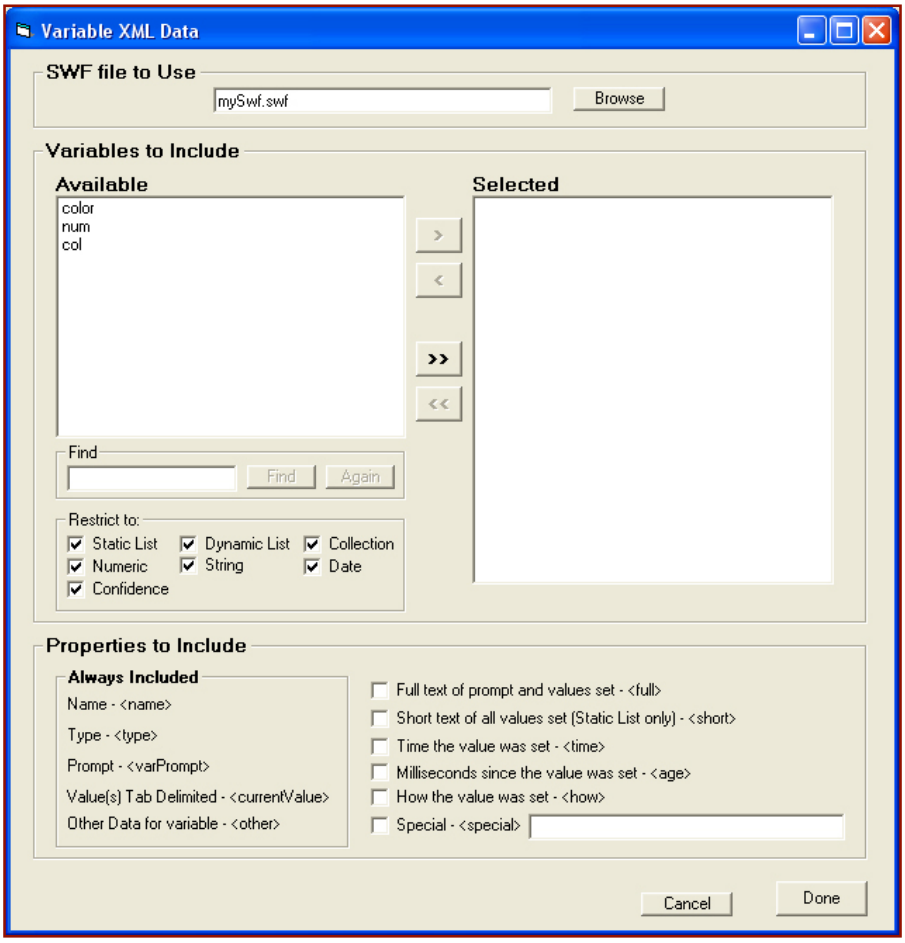
The “Display HTML page” edit box works essentially the same as for non-Flash systems, but when the Corvid Servlet Runtime is running in Flash mode, the command will automatically cause the runtime to send XML data with the action type tag set to “DISPLAY_HTML” and the action target tag set to the URL specified.

The other change on the Results tab is to allow specifying the variables to be included in the XML data for the results, and to specify what data will be sent for each variable.

The default and custom results commands now each have an edit box for “Flash Data To Send”. The information on what to send is stored in special files, which can be easily created or edited from within Corvid. The file name will be added to the Corvid Command as “FlashXML=*filename*”.

When adding a file for the Flash Results screen, either click “New” to create a new one, “Browse” to find an existing one, or “Edit” to edit the one currently entered in the edit box. This will open a window for specifying which variables are to be included in the XML sent to Flash when the command is executed.

The window for adding or editing the XML specification is:



1. Specify the SWF file to use at the top of the window. This can be any SWF file. Normally the SWF file(s) will be in the same folder as the Corvid CVR file and other system files. This makes paths unnecessary, and makes it far easier to move all the required files to the server.

2. Select the variables to include. All the variables in the system will be in the left list. You can limit the variables displayed in the left list using the “Restrict to” check boxes to display only certain types of variables, or use the “Find” to find variables with the search text in them. Select the variables to include and click the “>” button to add them to the “Selected” list. All the variables can be moved to the “Selected” list by clicking the “>>” button. Variables can be removed from the “Selected” list by clicking on them and clicking the “<“ button or the “<<“ button to remove all the variables in the “Selected” list.

3. Select which items of data on the variables are to be included in the XML. The <name>, <type>, <varPrompt>, <currentValue> and <other> values are always sent. The optional tags allow other information on the variable to be sent. Items should only be included if they will be used in some way in the Results display.

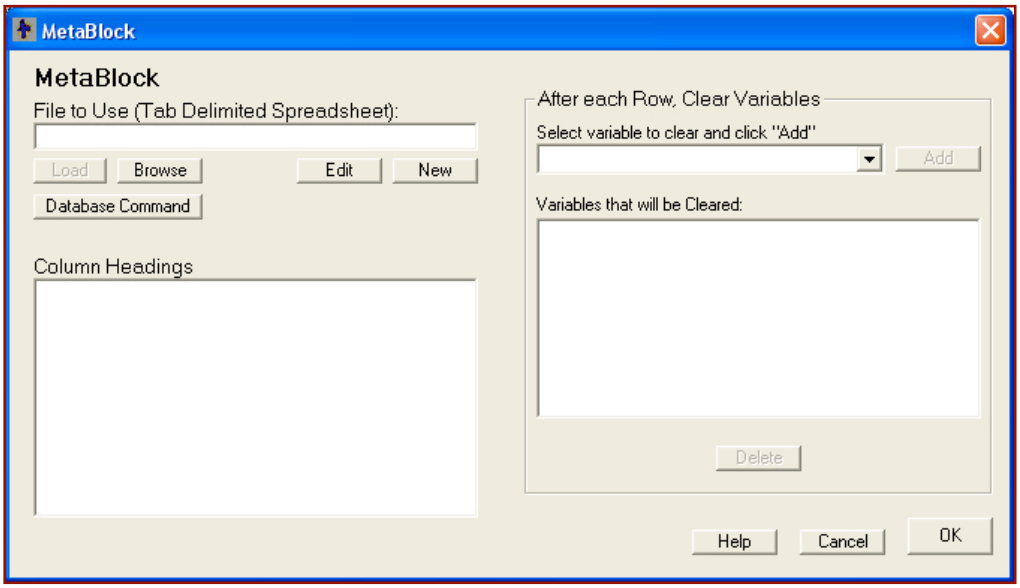
<full>	The full text of the prompt and any values set
<short>	The short text of any values set for Static List variables
<time>	The time the value was set
<age>	The time in milliseconds since the value was set
<how>	A detailed explanation of how the value was set
<special>	Any other string you would like to add - similar to <other>

The same tags will be sent for all the selected variables.

4. When done, if this is a new file, Corvid will ask you to specify a name for this XML specification file. Save it with your system files (CVR file) and move it to the server in the same folder as the CVR file when the system is fielded. When the system is run, the RESULTS will have and <exsysVar> tag for each of the selected variables, with the tags specified.

New MetaBlock Editor

In Corvid v5, the tab delimited files used in MetaBlock systems can be edited from within Corvid. Clicking on the “MetaBlock” button in a Logic Block window will display the window for setting the MetaBlock parameters.

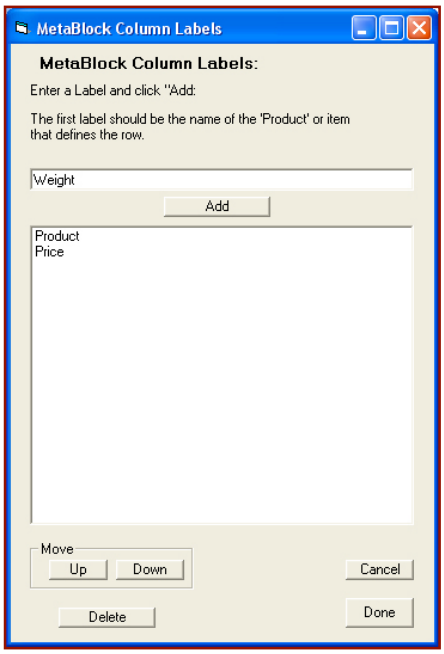


This window has 2 new buttons - “Edit” and “New” in the upper left.

To edit an existing tab delimited file, enter the file name in the “File to Use” edit box or click “Browse” and select the file. Then click the “Edit” button to open the file for editing.

Creating a New MetaBlock Data File

Clicking the “New” button will create a new tab delimited file that can be used for the data in a MetaBlock system. When a new file is created, Corvid will first display a dialog and ask you to select the name for the new file. It then displays a window to enter the labels to use for the columns. At this point, only the column headers are entered to establish the structure of the file. The actual data will be added later.

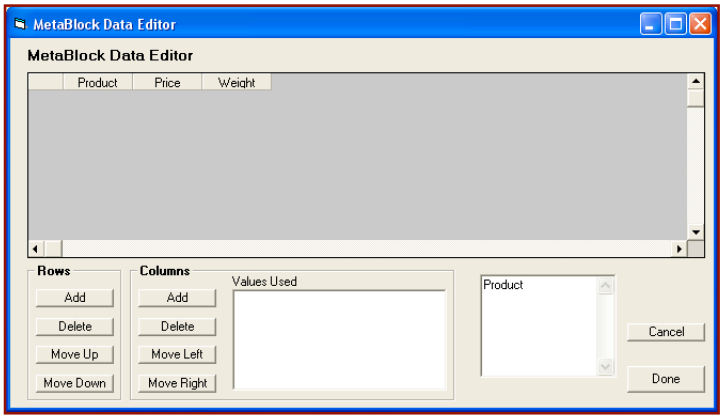


Enter the column label in the text box at the top of the window and click the “Add” button or press the “Return” key. The new label will be added to the end of the list.

Normally, the first column header label is the one that will identify the “products” that the MetaBlock system will select among. This is not required, but will make it much easier to build and maintain the data. Other columns can be in any order preferred. The column order does not matter to the logic in the Logic Block that will use the MetaBlock.

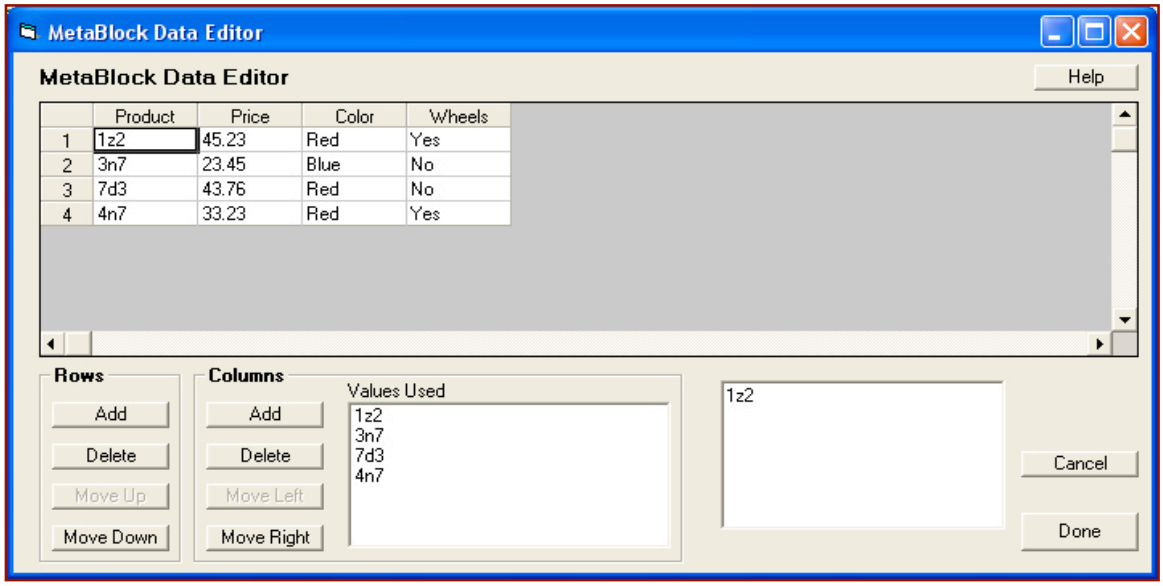
Items in the list can be moved or deleted by selecting the item and clicking the “Up” or “Down” buttons to move it and “Delete” to remove it. Additional columns can also be added later or moved after the file is created.

Once the column headers are entered, click “Done” to create the new file. When it first opens, it will only have the column headers.



Editing the Data

The Data Editor window displays a standard spreadsheet at the top, with controls for adding rows, columns and data below.



The column headers are displayed at the top. Rows are numbered for easy reference, but the numbers are not part of the file that will be output.

The width of columns can be adjusted by clicking on the vertical bar between headers and dragging to change the width. If the spreadsheet it too large for the window, it can be scrolled with the horizontal and vertical scroll bars.

Rows can be added to the spreadsheet by clicking the “Add” button in the “Rows” control group. This will add a blank row at the bottom of the spreadsheet. When creating a new file, click the “Add” button to add the first row of data.

A row can be deleted by clicking on any cell in a row to select it, and clicking the “Delete” button in the “Rows” control group.

Row order can be changed by selecting any cell in a row and clicking the “Move Up” or “Move Down” buttons.

Likewise, columns can be added by clicking the “Add” button in the “Columns” control group. A column can be deleted by selecting any cell in the column and clicking the “Delete” button. Columns can be moved left or right by selecting any cell in a column and clicking the “Move Left” or “Move Right” buttons.

The text in a cell can be edited several ways.

- The text in a cell can be directly edited by selecting it and directly changing the text

- Whenever a cell is selected, the text is also displayed in the edit box in the lower right. Editing the text in this box will automatically make the same change in the cell. This edit box also provides spell checking. It is easier to edit long text strings in this edit box.

- Whenever a cell in selected, all of the unique values used in that column are displayed in the "Values Used" list box. Double clicking on a value in that list box will copy the text into the cell. This is done to simplify building systems where a column should only have a limited set of possible values (e.g. Yes/No or red/blue/green). Once the values are entered once in a column, they can be easily copied. Also, the list box will quickly show if there are any incorrect values that might confuse the system. Columns where the value will be compared against a Static List variable value, or other fixed set of values, should use this to make sure that all values in the column are correct.

Once all of the data in the cells is edited, click the "Done" button and the data will be written out to the file. Clicking "Cancel" will exit the window and the original file will be unchanged.

New Applet Help System

Corvid v5 also makes use of a new Corvid application to help people that have problems running Java applets on their browser. Corvid systems distributed using the Corvid Applet Runtime will now display an HTML link in the applet window if the user's browser does not have the required software to run applets, or has the browser preferences set to block them. This link will take the user to a Corvid system that can diagnose the problem and tell them how to correct it. The system runs using the Corvid Servlet Runtime, so it will work even if applets are blocked. It can diagnose the cause for applets not running for all common browsers on MS Windows, Mac OSX and Linux, and provides detailed instructions on how to correct the problem. Corvid v5 automatically adds the link to this system when it builds applet tags. This will automatically be added to the default page Corvid builds to run systems or wherever the CORVID_RUNTIME_APPLET replaceable parameter is used. The link can also be easily added to existing systems.

You can run the applet help system by going to: **http://www.exsys.com/applethelp.html**

(Anyone using Java applets on their web pages is also welcome to use this system to assist their users in running other Java applets. Please use the above URL, which then links to the actual system. The location of the actual system may change, but the above link will be maintained.)

EXSYS
Capture Knowledge. Deliver Answers

Exsys, Inc.
6301 Indian School Rd. NE, Suite 700
Albuquerque, NM 87110 USA
(505) 888-9494, (505) 888-9509 *fax*

info@exsys.com
www.exsys.com

For support questions contact: **support@exsys.com**